CS-GA 2565 Final Course Project
Movie Rating Predictor

Anuva Sehgal
as18774@nyu.edu

December 23, 2023

1 Abstract

Machine Learning applications are multi-faceted and they can be used in different industries to extract meaningful insights. Predicting the success of a movie and the features important for it to reach that success can make or break a production's financial prospects. In this paper, we build a movie rating system and delve deeper into how some features impact those ratings more than others. We apply various data pre-processing techniques to prepare a clean dataset for our models. Some of the techniques covered here will be TF-IDF (Term Frequency-Inverse Document Frequency) and TruncatedSVD (Truncated Singular Value Decomposition). These two methods are used to numerically represent textual data and further reduce the dimensionality during feature extraction. We apply various regression models to our processed data and measure the success of each model using these metrics: MSE (Mean-Squared-Error), R^2 Score, and MAE (Mean-Absolute-Error). We then choose the best-performing algorithm and conduct additional exploratory analysis for feature importance and further comparisons. Finally, we summarize our findings through various visualizations and graphs.

2 Introduction

The motivation for this project comes from the very "buzzing" Over-the-top (OTT) media service industry. In this day and age of media consumption, various movie streaming platforms like Netflix, Amazon Prime, etc. are not only showing movies by way of acquiring exclusive rights but also producing many movies which is what they call "Originals". For that purpose, there is a decent amount of research and analysis that goes into creating and budgeting for in-house movies. This project attempts to capture what's behind the scenes of that process, by using the TMDB-5000 movie dataset and extracting meaningful insights into what aspects affect movie ratings and how much.

As mentioned before we use the TMDB-5000 movie dataset[1], as part of which we have 2 .csv files: tmdb_5000_movies.csv and tmdb_5000_credits.csv. The movies.csv file contains information about 5000 movies: budget, revenue, run-time, popularity, genres, etc. The credits.csv file contains information about the cast/crew of the same 5000 movies. For this project, we merge these two datasets (on movie_id) and use the combined result as our data frame going forward. The following snippets introduce the data and by extension the problem statement.

	budget	genres	homepage	movie_id	keywords	original_language	original_title	overview	popularity
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615

Figure 1: movies.csv

production_companies	...	runtime	spoken_languages	status	tagline	title_x	vote_average	vote_count
[{"name": "Ingenious Film Partners", "id": 289...	...	162.0	[{"iso_639_1": "en", "name": "English"}, {"iso...	Released	Enter the World of Pandora.	Avatar	7.2	11800
[{"name": "Walt Disney Pictures", "id": 2}, {"...	...	169.0	[{"iso_639_1": "en", "name": "English"}]	Released	At the end of the world, the adventure begins.	Pirates of the Caribbean: At World's End	6.9	4500

Figure 2: movies.csv

title_y	cast	crew
Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...

Figure 3: credits.csv

```
1 merged_df.at[1, 'cast']
```


```
'[{"cast_id": 4, "character": "Captain Jack Sparrow", "credit_id": "52fe4232c3a36847f800b50d", "gender": 2, "id": 8
5, "name": "Johnny Depp", "order": 0}, {"cast_id": 5, "character": "Will Turner", "credit_id": "52fe4232c3a36847f80
0b511", "gender": 2, "id": 114, "name": "Orlando Bloom", "order": 1}, {"cast_id": 6, "character": "Elizabeth Swan
n", "credit_id": "52fe4232c3a36847f800b515", "gender": 1, "id": 116, "name": "Keira Knightley", "order": 2}, {"cast
_id": 12, "character": "William \\"Bootstrap Bill\\" Turner", "credit_id": "52fe4232c3a36847f800b52d", "gender": 2,
"id": 1640, "name": "Stellan Skarsg\\u00e5rd", "order": 3}, {"cast_id": 10, "character": "Captain Sao Feng", "credi
t_id": "52fe4232c3a36847f800b525", "gender": 2, "id": 1619, "name": "Chow Yun-fat", "order": 4}, {"cast_id": 9, "ch
aracter": "Captain Davy Jones", "credit_id": "52fe4232c3a36847f800b521", "gender": 2, "id": 2440, "name": "Bill Nig
hy", "order": 5}, {"cast_id": 7, "character": "Captain Hector Barbossa", "credit_id": "52fe4232c3a36847f800b519",
"gender": 2, "id": 118, "name": "Geoffrey Rush", "order": 6}, {"cast_id": 14, "character": "Admiral James Norringto
n", "credit_id": "52fe4232c3a36847f800b535", "gender": 2, "id": 1709, "name": "Jack Davenport", "order": 7}, {"cast
_id": 13, "character": "Joshamee Gibbs", "credit_id": "52fe4232c3a36847f800b531", "gender": 2, "id": 2449, "name":
"Kevin McNally", "order": 8}, {"cast_id": 11, "character": "Lord Cutler Beckett", "credit_id": "52fe4232c3a36847f80
0b529", "gender": 2, "id": 2441, "name": "Tom Hollander", "order": 9}, {"cast_id": 19, "character": "Tia Dalma", "c
```


Figure 4: JSON formatted "cast"

```
1 merged_df.at[1, 'crew']
```


```
'[{"credit_id": "52fe4232c3a36847f800b579", "department": "Camera", "gender": 2, "id": 120, "job": "Director of Pho
tography", "name": "Dariusz Wolski"}, {"credit_id": "52fe4232c3a36847f800b4fd", "department": "Directing", "gende
r": 2, "id": 1704, "job": "Director", "name": "Gore Verbinski"}, {"credit_id": "52fe4232c3a36847f800b54f", "departm
ent": "Production", "gender": 2, "id": 770, "job": "Producer", "name": "Jerry Bruckheimer"}, {"credit_id": "52fe423
2c3a36847f800b503", "department": "Writing", "gender": 2, "id": 1705, "job": "Screenplay", "name": "Ted Elliott"},
{"credit_id": "52fe4232c3a36847f800b509", "department": "Writing", "gender": 2, "id": 1706, "job": "Screenplay", "n
ame": "Terry Rossio"}, {"credit_id": "52fe4232c3a36847f800b57f", "department": "Editing", "gender": 0, "id": 1721,
"job": "Editor", "name": "Stephen E. Rivkin"}, {"credit_id": "52fe4232c3a36847f800b585", "department": "Editing",
"gender": 2, "id": 1722, "job": "Editor", "name": "Craig Wood"}, {"credit_id": "52fe4232c3a36847f800b573", "departm
ent": "Sound", "gender": 2, "id": 947, "job": "Original Music Composer", "name": "Hans Zimmer"}, {"credit_id": "52f
e4232c3a36847f800b555", "department": "Production", "gender": 2, "id": 2444, "job": "Executive Producer", "name":
"Mike Stenson"}, {"credit_id": "52fe4232c3a36847f800b561", "department": "Production", "gender": 2, "id": 2445, "jo
b": "Producer", "name": "Eric McLeod"}, {"credit_id": "52fe4232c3a36847f800b55b", "department": "Production", "gend
er": 2, "id": 2446, "job": "Producer", "name": "Chad Oman"}, {"credit_id": "52fe4232c3a36847f800b567", "departmen
t": "Production", "gender": 0, "id": 2447, "job": "Producer", "name": "Peter Kohn"}, {"credit_id": "52fe4232c3a3684
7f800b56d", "department": "Production", "gender": 0, "id": 2448, "job": "Producer", "name": "Pat Sandston"}, {"cred
it_id": "52fe4232c3a36847f800b58b", "department": "Production", "gender": 1, "id": 2215, "job": "Casting", "name":
"Denise Chamian"}, {"credit_id": "52fe4232c3a36847f800b597", "department": "Art", "gender": 2, "id": 1226, "job":
"Production Design", "name": "Rick Heinrichs"}, {"credit_id": "52fe4232c3a36847f800b59d", "department": "Art", "gen
```


Figure 5: JSON formatted "crew"

```
1 merged_df.at[1, 'production_companies']
```


```
'[{"name": "Walt Disney Pictures", "id": 2}, {"name": "Jerry Bruckheimer Films", "id": 130}, {"name": "Second Mate
Productions", "id": 19936}]'
```


Figure 6: JSON formatted "production_companies"

```
1 english_movies_df.at[1, 'keywords']
```


```
'[{"id": 270, "name": "ocean"}, {"id": 726, "name": "drug abuse"}, {"id": 911, "name": "exotic island"}, {"id": 131
9, "name": "east india trading company"}, {"id": 2038, "name": "love of one\'s life"}, {"id": 2052, "name": "traito
r"}, {"id": 2580, "name": "shipwreck"}, {"id": 2660, "name": "strong woman"}, {"id": 3799, "name": "ship"}, {"id":
5740, "name": "alliance"}, {"id": 5941, "name": "calypso"}, {"id": 6155, "name": "afterlife"}, {"id": 6211, "name":
"fighter"}, {"id": 12988, "name": "pirate"}, {"id": 157186, "name": "swashbuckler"}, {"id": 179430, "name": "afterc
reditsstinger"}]'
```


Figure 7: JSON formatted "keywords"

As can be seen from the evidence provided above, this data was not directly usable for our models. In this project, not only do we parse JSON strings to extract useful fields but also drop some of the columns above to reduce noise and create new features using existing raw data (Feature Extraction). Hence, data pre-processing was a crucial element for this project, upon which our regression models perform training/testing.

Historically speaking, there has been quite a bit of work done in this space. Some projects use the same dataset to address different problem statements and some projects use a different dataset to approach a similar problem. For example, one project uses the IMDb dataset to predict movie ratings of new movies[2], and another one, Predict Movie Ratings via Machine Learning[3] uses the same dataset for predicting the movie ratings but their inference and approach are different from what we do in this paper.

3 Related Work

There is abundant literature available on most of the applications of machine learning. The topic for this paper is no different. Some projects tie closely with ours in terms of the dataset and intermediate objectives while others hold similarity in inspiration. Two of those projects are presented in the "Introduction". Another similar project is the Rotten Tomatoes Movie Rating Prediction[4]. This project uses the rotten tomatoes dataset to predict the success of a movie. It attempts classification and sentiment analysis in its approach. Again, the motivation behind this project and the one described is shared but we have maximized our efforts to stay as original as possible amid the plethora of resources.

4 Method

For the sake of structure, we will address the dataset in this section, so we can smoothly transition into the applied ML algorithms.

4.1 Dataset

As discussed in the introduction, a crucial part of this project was the data pre-processing. Let's go through them step-by-step:

4.1.1 Filtering

Following are the columns of our merged dataset:

1. budget: budget of the movie in USD
2. genres: one movie can have multiple genres e.g. [Adventure, Fantasy, Action]
3. homepage: homepage of movie
4. movie_id: preset unique identifier
5. keywords: descriptive words distinguishing the movie e.g. snippet given for Pirates of the Caribbean (Figure 7)

6. original_language: the original language of the movie e.g. en (English), fr (French), etc.
7. original_title: the title of the movie e.g. Pirates of the Caribbean: At World's End
8. overview: plot summary of the movie
9. popularity: the popularity is based on a couple of factors. This metric is provided by TMDB[5]. It factors in total votes, release date, etc.
10. production_companies: companies that produced the movie. e.g. refer to Figure 6 for Pirates of the Caribbean
11. production_countries: the countries associated with the movie's production companies[6]
12. release_date: the release date of the movie e.g. 2016-07-29
13. revenue: revenue collected by the film in USD
14. runtime: movie runtime in minutes
15. spoken_languages: all languages spoken in a movie e.g. French, English, Italian, Spanish, Dutch in Spectre
16. status: released or not released
17. tagline: tagline of the movie e.g. At the end of the world, the adventure begins for Pirates of the Caribbean: At World's End
18. title: title of the movie
19. vote_average: total voter rating/number of people who voted
20. vote_count: number of people who voted
21. cast: actor names with their character names and other IDs e.g. Figure 4
22. crew: crew member names with their job, department, gender, etc. e.g. Figure 5

Some of these columns can be dropped e.g.: 'homepage', 'movie_id' (not needed for our project), 'production_countries', 'spoken_languages', 'status', and 'original_title'. It becomes obvious because it can be seen by the common eye that none of these columns give any useful information for the objective that this project tries to achieve.

The next step in filtering unnecessary columns is to focus our efforts on English movies. There are 298 non-English movies, including French, Spanish, Dutch, Chinese, Japanese, Portuguese, and more. Since we have a small dataset for each language individually, it's better to make good predictions on the language we have available than to make bad predictions using languages that

aren't appropriately represented in the dataset. Next up, some columns like Cast, Crew, Production_companies, and Keywords were in a JSON formatted string (refer to Fig 4-7). A sample JSON string looks like this:


```
{
   "user": [
       {
           "id": 1,
           "gender": "Female",
           "first_name": "Susan",
           "last_name": "Huetson",
           "email": "shuetson0@amazon.de",
           "ip_address": "47.47.39.223",
           "friend": [
               {
                "first_name": "Querida",
                "last_name": "Clark"
               }
           ]
       }
   ],
   "comment_id": 11111,
   "comment": "This is my first comment!",
   "post_id": 99999
```


Figure 8: Sample JSON

Each element in these columns needed to be parsed and the "name" field extracted. This "name" field for Actors would give us the name of the actor playing a character. For keywords, it would give us the actual content of the keyword (stripping away the IDs, etc.), similarly for genres, overview, and production_companies. When we process all this data we replace the respective column entries with the new "clean" data. Now, for the 'crew' column, it was a different story. The JSON string for that consisted of not just the IDs, names, etc. but also the job and department. For the sake of simplicity, we just focus on the "director" for each film, so we parse each movie crew object for the director and populate a new column for that. The following gives a peek into how these processed columns look like:

genres	keywords	original_language	overview	popularity	production_companies	release_date	revenue	runtime	tagline	title
[Action, Adventure, Fantasy, Science Fiction]	[culture clash, future, space war, space colon...	en	[In, the, 22nd, century,, a, paraplegic, Marin...	150.437577	[Ingenious Film Partners, Twentieth Century Fo...	2009-12-10	2787965087	162.0	[Enter, the, World, of, Pandora.]	Avatar

Figure 9: Cleaned Genres, Keywords, production_companies, and split Overview and tagline

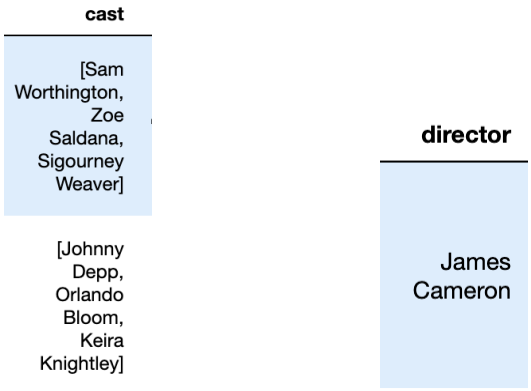

Figure 10: Cleaned cast and crew

Notice here in Figure 9, that we converted columns like overview and tagline from sentences to one word 'tags' so we can later combine all string literals into one 'tags' column for further text-to-numerical conversion.

While parsing actors, we only extract the top 3 leading actors in the film. Fortunately, the JSON string is formatted in such a way that the order of the objects is in the order of the significance of the character's role in the movie. Previously we extracted the top 5 actors keeping in mind, movies like The Avengers where the cast is comprised of highly rated actors who play equally significant roles. On manually checking the dataset we saw that there weren't enough movies with a large cast like The Avengers, so it didn't give us much confidence in making the trade-off between noise and accuracy for a small number of movies.

Whilst doing all this preliminary processing, we dropped all rows with NaN director values, or where all 3 actors are missing. After the final data preprocessing and just before applying Standardization/Normalization, we double-check for NaNs and drop any rows with missing values. Fortunately, there is only 1 such row left after all the pre-processing.

for crew JSON strings. Similarly, Genres needed to be parsed but the 20 unique genres spanned across the movies data needed to be one-hot label encoded.

4.1.2 Feature Extraction

Firstly, for the filtered actor column, we create 3 new columns: Actor_1, Actor_2, and Actor_3, and populate them with their respective values from the parsed list of 'cast'. We then drop the 'cast' column altogether and move on with our analysis.

We now reach the point of the project where it starts getting interesting. At this point, we have text data like 'Johnny Depp', 'Orlando Bloom', 'Daniel Craig', etc. (actors) or 'Christopher Nolan', 'Steven Spielberg', etc. (directors) or 'Action', 'Horror', 'Comedy', etc. (genres) and 'Warner Brothers', 'Walt

Disney Pictures', etc. (production_companies). For us to make the machine learning model understand what these fields and values mean, we needed to numerically represent these values in a way that maintained integrity and conveyed the importance of the name. For this purpose, we create a new metric for Actor_1, Actor_2, Actor_3, Director and production_companies. This metric will be called 'popularity'[7].

The way we calculate Actor_1_popularity, Actor_2_popularity, Actor_3_popularity, and director_popularity is by searching for all movies that actor x or director y is part of, then calculating the popularity of each of that movie: $movie['vote\_average'] * movie['vote\_count']$ and taking a sum. In short, the popularity of actor x will be a sum of the above product over all the movies done by that actor (in the dataset). Similarly for director_popularity. NOTE: We could not have used the column 'popularity' itself because that column singularly didn't give a holistic view of the ratings. Similarly, for production_companies we used the same formula but since the column contained a list of companies (one movie can be produced by multiple production_companies) we added another layer of iteration. Following are the top 10 results for each :

```
# Display the top 10 items
print(top_10_actors)
```

```
{'Leonardo DiCaprio': 573244.2000000001, 'Robert Downey Jr.': 485870.9000000001, 'Brad Pitt': 479033.80000000005,
'Tom Hanks': 471864.99999999994, 'Johnny Depp': 426737.69999999984, 'Ian McKellen': 418728.5999999999, 'Christian B
ale': 398624.49999999994, 'Matt Damon': 396771.0999999999, 'Will Smith': 345295.60000000003, 'Harrison Ford': 33662
0.30000000005}
```

Figure 11: Top 10 actors according to popularity score

```
# Display the top 10 items
print(top_10_directors)

```

```
{'Christopher Nolan': 497789.60000000003, 'Steven Spielberg': 431883.2999999999, 'Peter Jackson': 345623.0, 'Quenti
n Tarantino': 340744.30000000005, 'David Fincher': 281189.7, 'Ridley Scott': 260574.8, 'Martin Scorsese': 253011.2,
'Robert Zemeckis': 247910.3, 'James Cameron': 242153.49999999997, 'Tim Burton': 194046.39999999997}
```

Figure 12: Top 10 directors according to popularity score

```
# Display the top 10 items
print(top_10_pc)
```

```
{'Warner Bros.': 2993639.4999999986, 'Universal Pictures': 2108854.6000000006, 'Paramount Pictures': 1947716.500000
0002, 'Twentieth Century Fox Film Corporation': 1925577.7999999998, 'Columbia Pictures': 1427589.6999999995, 'Walt
Disney Pictures': 1185488.5999999992, 'New Line Cinema': 1056761.9999999998, 'Legendary Pictures': 849804.999999999
9, 'Village Roadshow Pictures': 785353.4999999997, 'Amblin Entertainment': 686083.8999999999}
```

Figure 13: Top 10 production companies according to popularity score

So we now create popularity columns for Actor_1, Actor_2, Actor_3, Director, and production_companies and drop the original text type columns. Now, for genres, we saw that we have 20 unique genres:

```
['TV Movie', 'Action', 'Drama', 'Comedy', 'Horror', 'Mystery', 'Fantasy', 'Music', 'Science Fiction', 'Romance', 'F
amily', 'War', 'Thriller', 'Adventure', 'Documentary', 'Crime', 'Foreign', 'History', 'Animation', 'Western']
```

Figure 14: Unique genres

Each movie can have multiple genres so to capture this multi-label categorical data for each movie, we used one-hot-encoding and created 20 new columns (one for each genre). We then set '1' for all genres of a particular movie and do that for all the movies. At the time of implementation, we didn't realise that we had performed one-hot-encoding so the code does not use the Sci-kit Learn Library. We do realize that this will make the data sparse by adding 20 new features (most of which will have 0 value for movies) but a decision was made and we moved forward with the processed data. Needless to say, we dropped the original 'genre' column after this.

Now comes the last bit of textual data that we need to tackle. The 'overview', 'tagline', 'title', and 'keywords'. All contain strings that in some shape or form describe the plot of the movie. For this reason, we combine all these columns under a common name: 'tags' and drop the rest of the original columns. Now these 'tags' will contain some rare, unique, and significant words e.g. alien, comic, family, etc., and yet some unnecessary words like in, the, a, an, of, etc. (also some punctuation marks: '.', '/', ',' etc.). We first file the unnecessary 'stopwords' using 2 libraries: NLTK and SpaCy. We use the common set of English words: words from NLTK and the language model: en_core_web_sm from SpaCy. The reason for using 2 different libraries was simply to achieve as much filtering of the tags as possible. On manually going through the data, we found that after using just NLTK there were still some common words that added no significance for our purpose. During this cleaning, we also used WordNetLemmatizer to group variant forms of the same word. E.g. happiness will turn to 'happy', meeting becomes 'meet', dogs becomes 'dog' etc. The final result for the tags is:

```
1 english_movies_df['tags'][1]
```

```
'Captain Barbossa long believed to be dead has come back to life and is headed to the edge of the Earth with Will T
urner and Elizabeth Swann But nothing is quite as it seemsocean drug abuse exotic island east india trading company
love of ones life traitor shipwreck strong woman ship alliance calypso afterlife fighter pirate swashbuckler afterc
reditsstingerAt the end of the world the adventure beginsPirates of the Caribbean: At Worlds End'
```

```
1 english_movies_df['filtered_tags_after_more_NLTK'][1]
```

```
'captain long dead come life headed edge earth turner drug abuse exotic island east trading company love life trait
or shipwreck strong woman ship alliance calypso afterlife fighter pirate swashbuckler end world adventure end'
```

Figure 15: NLTK and SpaCy conversion result

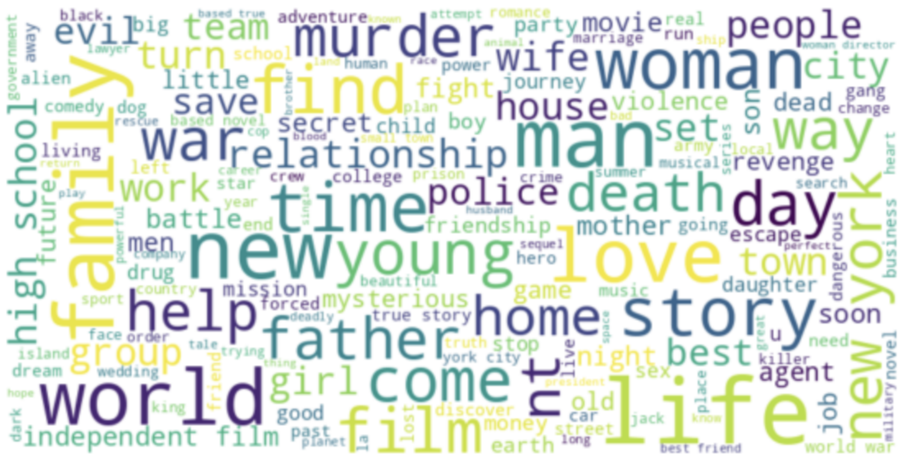

Figure 16: WordCloud for the tags

Now that we have essential words as our 'tags' we can continue to convert the text into a numerical representation. For this purpose, we used the TF-IDF[8] (Term Frequency- Inverse Document Frequency) model. The reason why we chose it over other text-to-numerical conversion b=models like Bag-of-words (studied in class) or Word2Vec etc. is because TF-IDF takes into account how often a word appears in a document (TF, ¿0) and how unique it is across documents (IDF, calculated as a log of number of documents in corpus/number of documents in corpus containing the term so ¿=0). For a movie plot, this emphasis on rare words was an obvious choice. Additionally, TF-IDF also produces relatively less sparse matrices than an approach like BoW. We get 1000 features from the TF-IDF matrix looking something like this:

```
print(tfidf_matrix[3])
print(tfidf_matrix.shape)
```

```
  (0, 487)      0.1465017421567553
  (0, 928)      0.16862097543087912
  (0, 193)      0.14133447812450198
  (0, 124)      0.1569717589226434
  (0, 227)      0.1696015282195891
  (0, 392)      0.1269157532542698
  (0, 900)      0.16336625315155004
  (0, 853)      0.14169974982428554
  (0, 947)      0.17390894326693115
  (0, 86)       0.15894501763265967
  (0, 887)      0.09413109540167745
  (0, 249)      0.13991934601453315
  (0, 408)      0.15454040983644357
  (0, 420)      0.12824661845137275
  (0, 192)      0.1250312491571342
  (0, 464)      0.3455365177633048
  (0, 205)      0.24807347079059747
  (0, 477)      0.14244493764244284
  (0, 879)      0.298504884000783
  (0, 588)      0.07702991581381362
  (0, 576)      0.11887708557458153
  (0, 472)      0.13728748329508206
  (0, 650)      0.10933617539706866
  (0, 145)      0.3207144789326065
  (0, 471)      0.15572964833842115
  (0, 679)      0.27915296840434417
  (0, 47)       0.1727682588816524
  (0, 241)      0.17509818176890876
  (0, 216)      0.10075435521423225
  (0, 768)      0.10399252065008618
  (0, 311)      0.15177152392829985
  (0, 325)      0.15230378549038584
(4454, 1000)
```

Figure 17: TF-IDF Matrix sneak peek

Now 1000 features look like a lot, but keep in mind that TF-IDF considers the most frequent terms, and the selected terms might not necessarily be the most informative or relevant for our specific task. Hence, adjusting max_features (parameter for TfidfVectorizer) is a trade-off between computational efficiency and the information retained in the representation. Therefore, to reduce dimensionality we employ a method called TruncatedSVD (Truncated Single Value Decomposition). This was an alternate approach to the original motivation to use PCA(Principal Component Analysis). Since PCA didn't work with the sparse matrix that TF-IDF outputted, our next option was to apply Truncated SVD[9] which is a Matrix Factorization technique. Using this technique we reduced the 1000 features down to 500 without losing the significance of tags.

```
Top features for SVD Component 1:
life: 0.1990
new: 0.1953
love: 0.1932
family: 0.1584
man: 0.1581
world: 0.1568
young: 0.1532
woman: 0.1463
story: 0.1444
film: 0.1423
```

```
Top features for SVD Component 2:
war: 0.4746
world: 0.3168
space: 0.1581
alien: 0.1509
mission: 0.1224
army: 0.1136
battle: 0.0966
earth: 0.0965
secret: 0.0939
planet: 0.0862
```

```
Top features for SVD Component 3:
school: 0.6322
high: 0.4347
teacher: 0.1065
girl: 0.0918
teen: 0.0872
group: 0.0857
student: 0.0801
party: 0.0742
college: 0.0741
killer: 0.0661
```

```
Top features for SVD Component 4:
new: 0.4297
york: 0.3950
city: 0.2316
police: 0.1576
murder: 0.1249
agent: 0.0917
cop: 0.0910
killer: 0.0888
drug: 0.0864
crime: 0.0716
```

Figure 18: Example "baskets" Truncated SVD puts the 'tags' in

To prove how effective TruncatedSVD was in reducing the dimensionality without losing much meaning, we plot the explained variance graphs as follows:

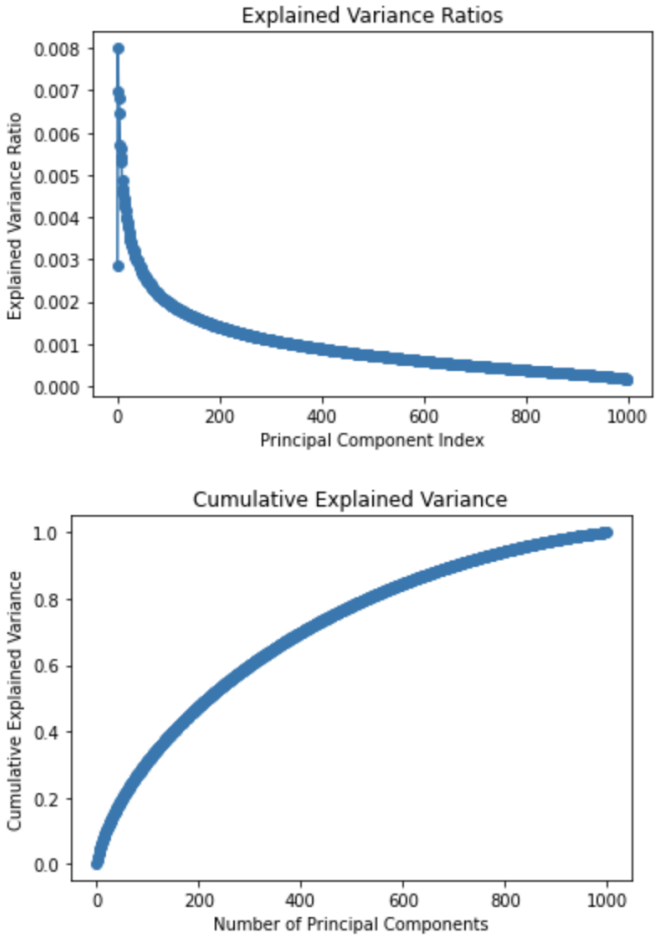

Figure 19: Explained Variance Graphs

Also, we can see that the reconstruction error of going back from (4454, 500) to (4454, 1000) matrix is 0.0002 which is quite seamless. Now we drop the 'tags' and use the new 500 features instead. With a total of 536 features, we need to make sure if there is anything else we can remove before moving ahead. For this purpose, we use a seaborn heatmap to plot correlations between features. This will help us to remove unnecessary columns that convey duplicate or redundant information. The following is the resultant plot:

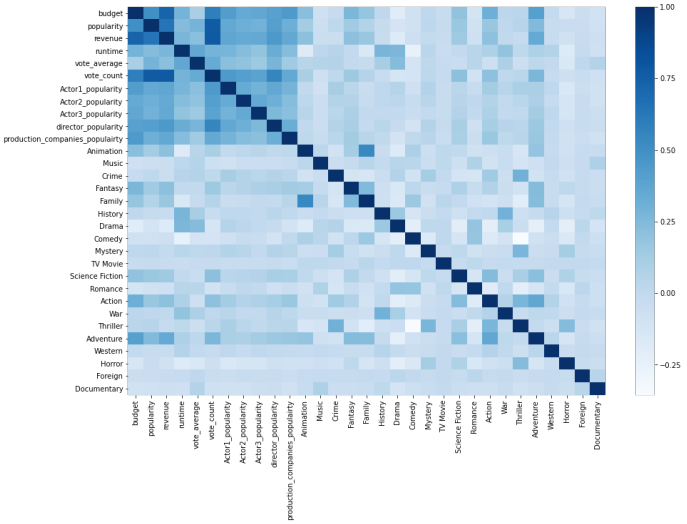

Figure 20: Heatmap of features excluding SVD components

Looking at this heatmap, we can see that vote_count is highly correlated to revenue and popularity. Also, the popularity, budget, and revenue "area" show a high positive correlation which means they are trying to convey the same thing. Hence, we drop the following columns: revenue, popularity, and vote_count. We plot the heatmap again to cross-check our decision and continue to the next step which is: Scaling.

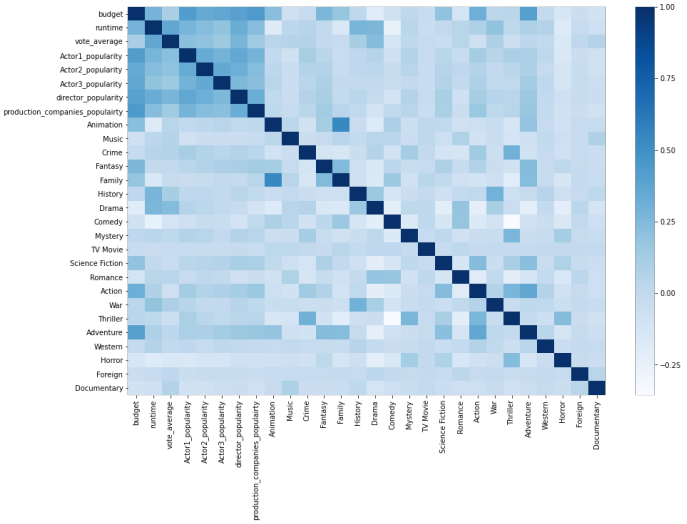

Figure 21: New heatmap with lesser correlation

Note: Our choice of dropping the columns above also aligns with the logical fact that popularity and revenue are things that are calculated after the movie is out and available for the audience to watch. Having these fields in the dataset would heavily bias the results and lead to overfitting and false predictions. Similarly, the vote_average is calculated using the vote_count itself so having vote_count will not only cause bias in the independent variables but also add duplicity of information.

4.1.3 Feature Scaling

This section of the pre-processing usually involves choosing between Standardization vs Normalization[10]. Standardization centers data around a mean of zero and a standard deviation of one(StandardScaler), while normalization scales data to a set range, often [0, 1], by using the minimum and maximum values(MinMaxScaler). Standardization will make the data less sensitive to outliers while Normalization ensures that no single feature disproportionately impacts the results. Normalization does not change the distribution of the data. It is recommended to apply Normalization to distributions that are unknown or "not" Gaussian. And vice versa for Standardization. Due to the large number of features in this dataset, we couldn't afford to skew or taint our carefully curated features, so we decided to study the distributions and apply a combination of the two to the features.

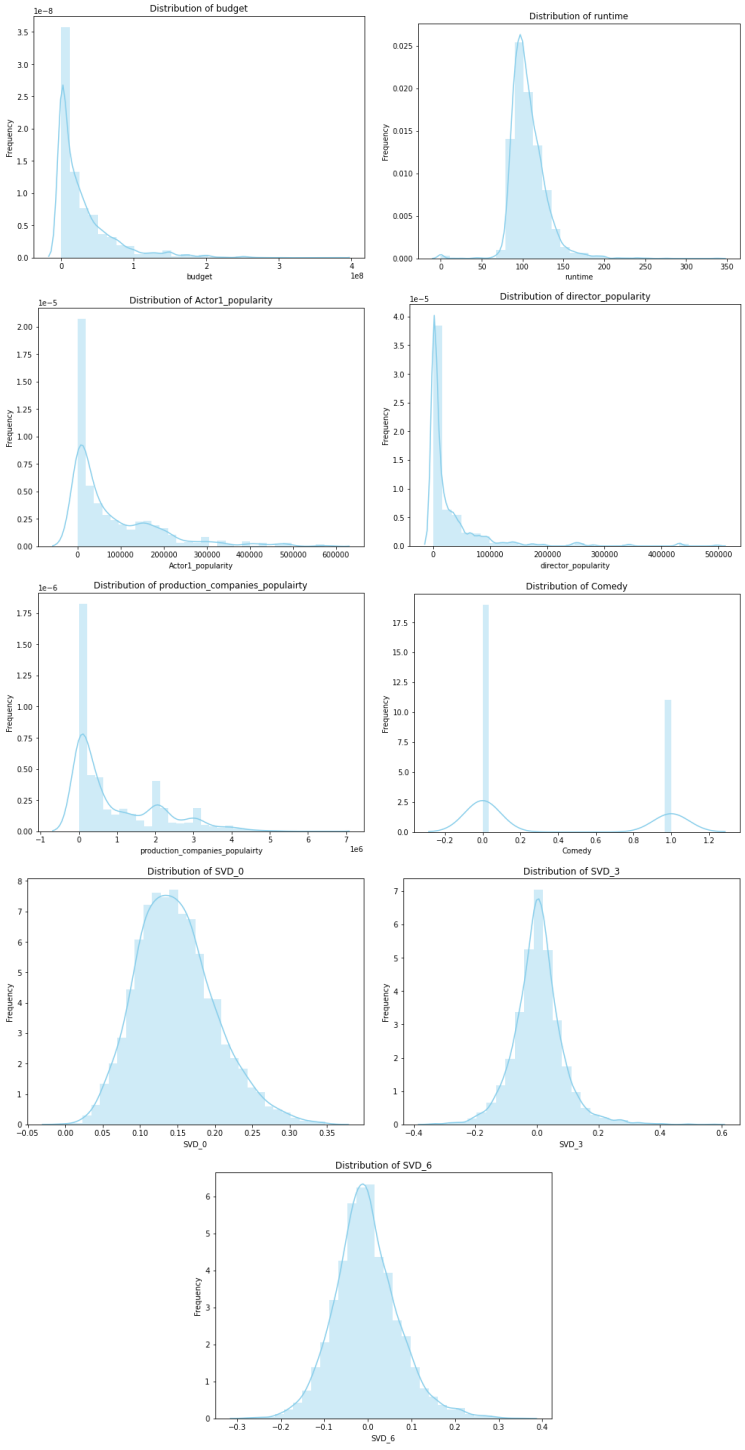

Figure 22: Feature distributions

As can be seen from the plots above, some features are not normally distributed. All genres will be a categorical 0 or 1, other popularity graphs have multiple peaks, budget, and runtime are highly skewed, etc. So, we apply Standardization to features with skewness ¡ 0.5 and Normalization to the rest. This can not only be understood by the model but now we create a common scale with minimum distortion to the distribution.

Now then, we extract the vote_average column from the data frame and use it as the 'y' output vector(true values). We then split the 'x' and 'y' arrays in a 70/30 training (x_train, y_train) and test set (x_test, y_test) and start regressions.

4.2 Regressions

We apply the following regression models to our data: Linear Regression, Ridge Regression, SVR: Support Vector Regression, Random Forest, and XGBoost. The performance metrics that stay consistent for all models are MSE (Mean Squared Error), $R*2$ Score, and MAE (Mean Absolute Error). MSE and MAE are straightforward and common terms. They measure how different the prediction is from the true value "on an average". The R*2 score lies between 0 and 1. The closer it is to 1, the better the predictions. It is:

$$R^2 = 1 - \frac{RSS}{TSS}$$

where: R^2 = coefficient of determination RSS = sum of squares of residuals, TSS = total sum of squares

4.2.1 Linear Regression

We start our analysis with the simplest model. We train the model using the sklearn library and run predictions. The performance is as follows:
Mean Squared Error: ***1.095****152865576995*
R-squared: ***0.137****87728640394836*
Mean Absolute Error: ***0.74****08417501407152*

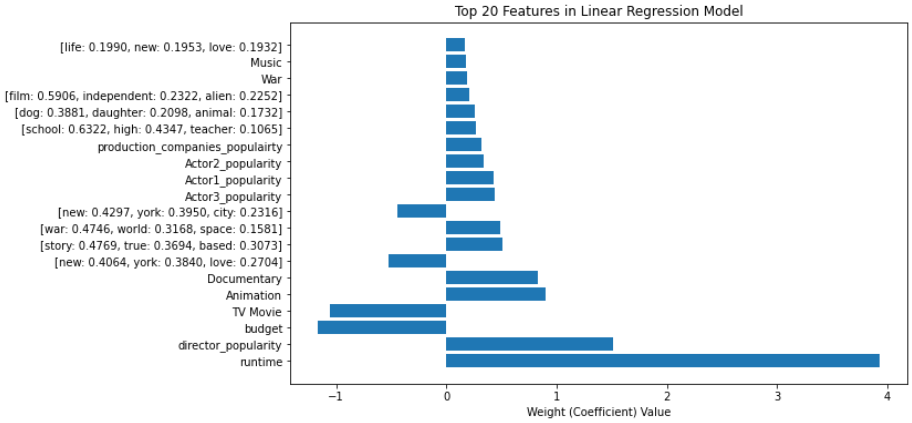

Figure 23: Feature importance from Linear Regression

Runtime and director_popularity have a heavy positive effect on the voter ratings, while budget had a negative effect. That means lengthier movies and movies directly by good directors did well at the box office. Also, the low-budget movies did surprisingly well. The feature importance graph also shows that some genres e.g.: Animation and Documentary did better than others. The plot also shows how Actor_3's popularity is more important than Actor_2 or Actor_1. If you think about it, it makes sense in a way that almost all movies will have a famous leading actress/actor but as you go to the $2nd$ and then the $3rd$ "leading" character, the popularity drops. So, if the $3rd$ actor is also famous, that means the movie cast line-up is quite prominent which leads to a better rating for the film. These insights will change with the change in model(not a stark difference!) but we will discuss more of such insights for the models that give better performance and are closer to the true values.

4.2.2 Ridge Regression

Running ridge regression with multiple regularizers (hyperparameter), λ : [0.00001, 0.001, 0.01, 0.1, 1.0, 10.00, 100], we see that the value of 10.0 gives the best result among all:

Mean Squared Error: ***1.1056****479430313197*
R-squared: ***0.129****61538531349548*
Mean Absolute Error: ***0.736****0157079102139*

Again, it is not the most performant model, but it performs slightly better than the liner model. Following is the feature importance graph:

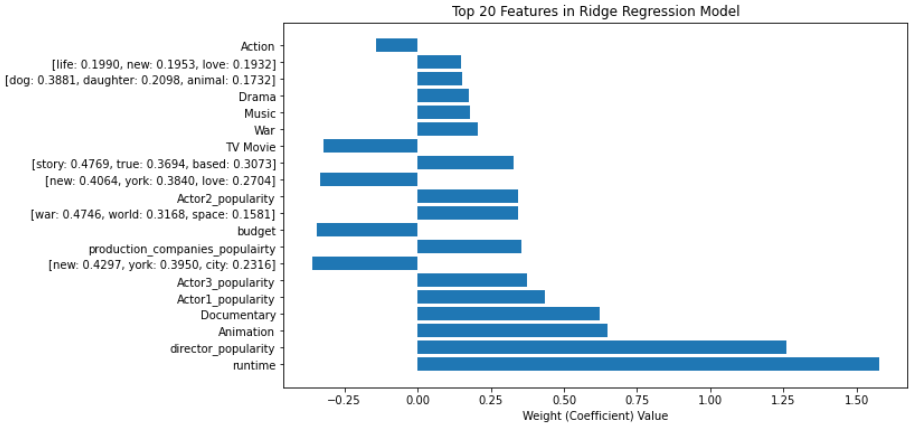

Figure 24: Feature importance from Ridge Regression

This plot is different from the one we obtained from linear regression. Note: these feature maps should be looked at from the bottom up. The difference we see here is that Actor_1 popularity is more important than Actor_3 popularity. While we can come up with an argument to support this insight, we'll leave it as is because we'll see in later models that Actor_3 popularity is more important. Other than that, there are tags like 'new', 'york', 'city', 'war', 'world', 'space', 'love', 'story', 'true', 'based', etc. that mostly contribute positively to the rating. For the ones with negative weights, we interpret that a positive effect in the y value, i.e. the rating, will result in a negative effect on the independent variable whose coefficient is negative. The same understanding can be applied to the 'tags' in linear regression.

4.2.3 Support Vector Regression (SVR)

We applied 3 types of kernels in the SVR model: linear, polynomial, and RBF. The latter two were employed in an attempt to capture any non-linearity in the data.

We trained the linear kernel with the 3 different regularization, C, hyperparameter: [0.1, 1, 5]. Other values for c¿5 were increasing the complexity of the model and hence resulting in longer processing times. So we stick with c=5. Greater values of C mean that the regularization is weak. This allows the model complexity to increase as the penalties decrease (C in SVR is the inverse of λ in ridge regression). We get similar results for c=1 and c=5 with the error (MAE) difference being just approx. 0.001.

linear SVR for c= 1
Mean Squared Error: ***1.0684****11768969308*
R-squared: ***0.158****9283263971707*
Mean Absolute Error: ***0.711****0824040814239*

linear SVR for c= 5
Mean Squared Error: ***1.067****4169075882003*
R-squared: ***0.159****71149797119621*
Mean Absolute Error: ***0.712****9361651087879*

Also note that for comparable predictions, we compare the MAE[11], as it is the most robust while MSE is more sensitive to outliers, and R^2 score, is the proportion of variance explained by the model. While R2 score is independent of context, the MAE will represent the loss in an absolute sense that has the same unit as the output variable.

For the polynomial kernel, we tried the following hyperparameters: C in [0.1, 1, 5] and degree in [2, 3, 4]. So not only did we try different regularizers but also different degrees of polynomials. The following parameters yielded the best results:

polynomial SVR for c= 1
polynomial SVR for degree= 2
Mean Squared Error: ***1.25****18020490947142*
R-squared: ***0.0145****60420401183882*
Mean Absolute Error: ***0.775****1412395516479*

We see that the polynomial kernel error is more than the linear kernel so it's safe to interpret that it might be overfitting the data (maybe even start fitting noise).

For the RBF kernel, we used the same λ=[0.1, 1.0, 5] as the linear kernel. The best performance we got was with c=1 and the results are as follows:

RBF SVR for c= 1
Mean Squared Error: ***1.1679****58880357713*
R-squared: ***0.080****56316980726963*
Mean Absolute Error: ***0.739****2145834674285*

Again, this performs worse than the linear kernel itself but still better than the polynomial kernel. This could be because the RBF kernel[12] addresses the issues of overfitting introduced by the polynomial kernel alone due to its complex yet efficient nature of combining multiple polynomial kernels of different degrees multiple times to project the non-linearly separable data into higher dimensional space. (see).

Overall, the SVR method didn't yield any exceptional results for us to stop our search here. So we move on to Random Forests.

4.2.4 Random Forest

Now, in the hopes of leveraging the aggregations done during the ensemble learning in Random Forest, we applied this technique with the hyperparameter:

n_estimators = 50. On trying out a couple of values 10-50, 50 gave us the best results. We also applied K-Fold cross-validation (k=5) with the model trained with 50 decision trees, which got a Mean MSE of 0.61 and Standard Deviation of MSE: 0.029 which confirms that this hyperparameter indeed outputs stable results. The model performance was as follows:

Mean Squared Error: ***0.57****19958083832335*
R-squared: ***0.549****7153009510609*
Mean Absolute Error: ***0.58****0940119760479*

This is by far the best results we've gotten and hence we do some more exploratory analysis on this model to extract insights.

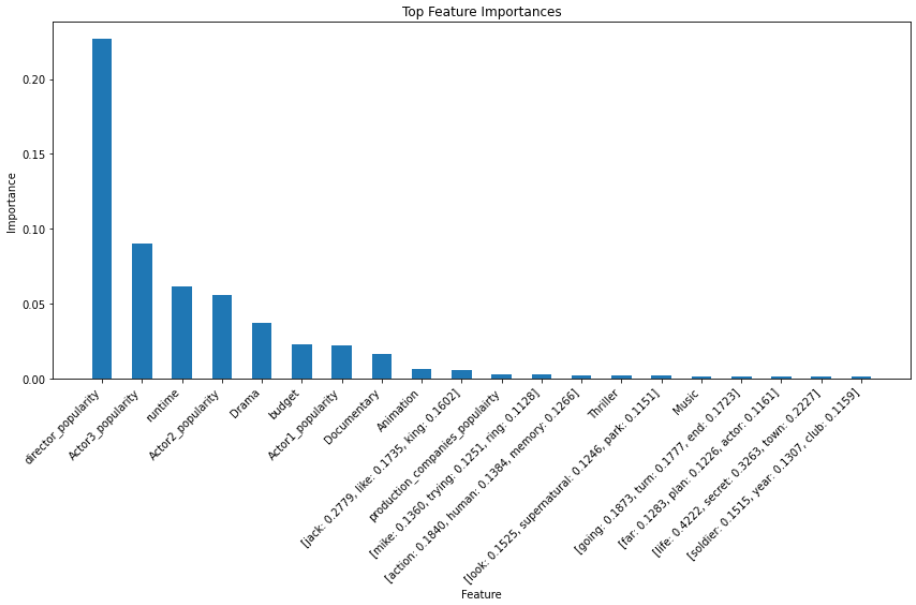

Figure 25: Top feature performances

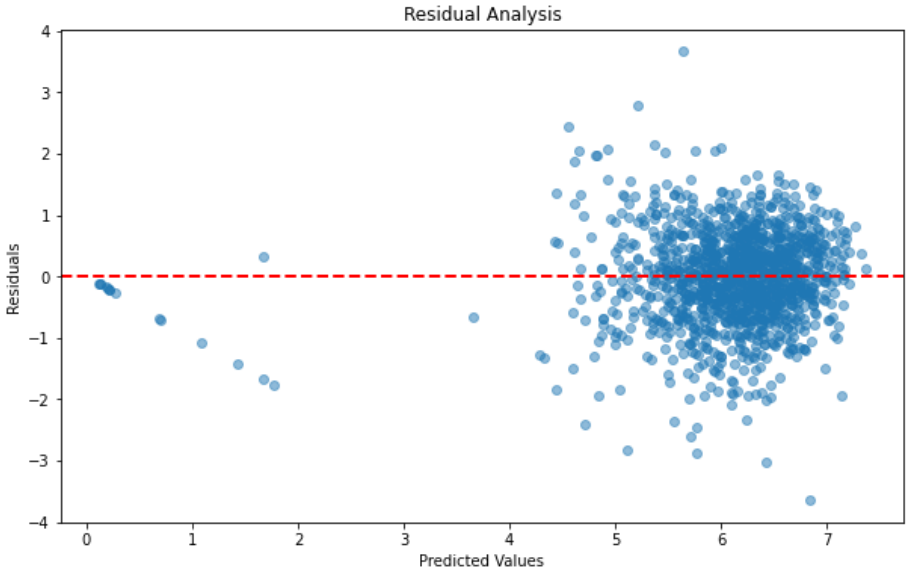

Figure 26: Residual analysis

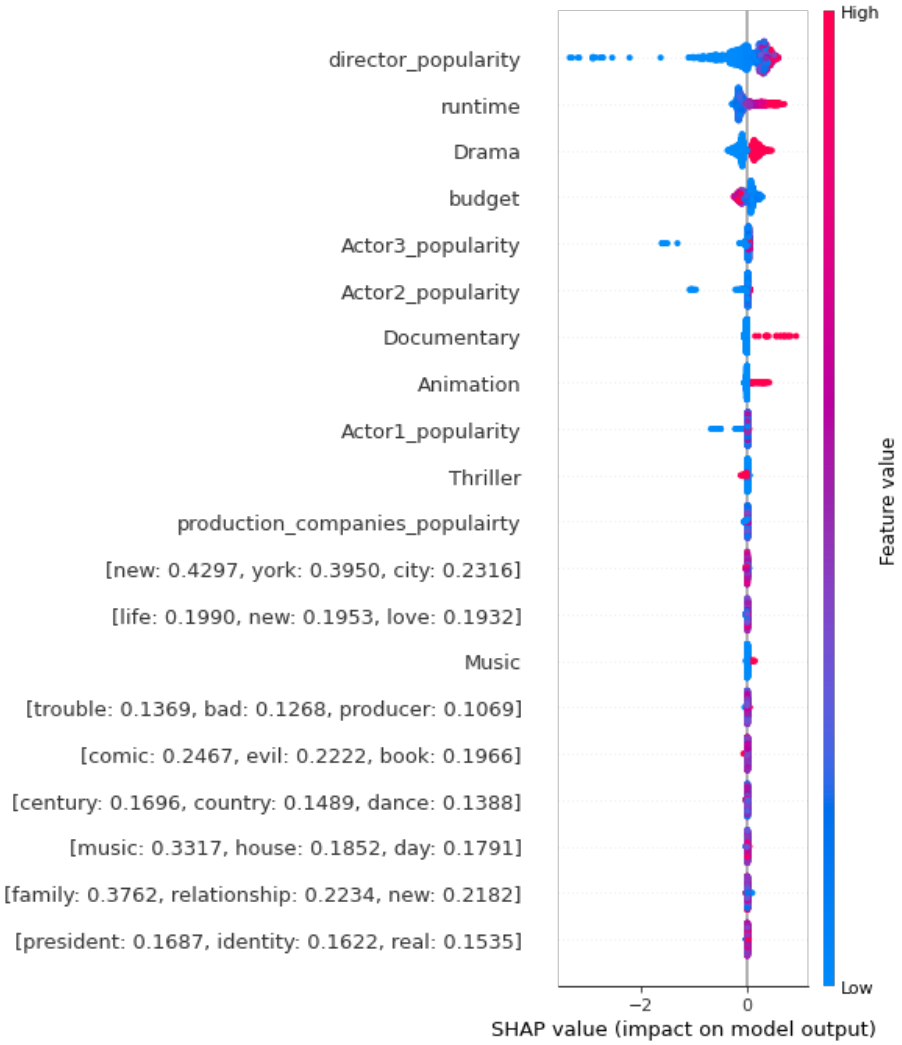

Figure 27: SHAP Summary Plot

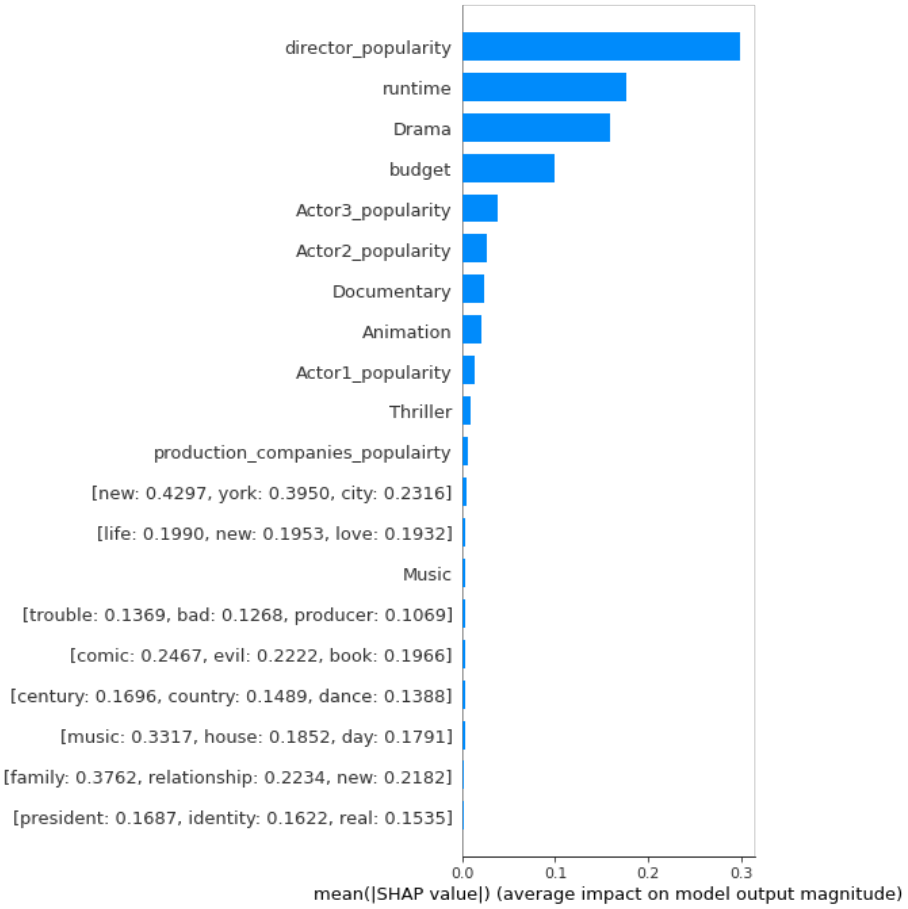

Figure 28: SHAP Feature importance

Figures 23 and 26 both show the feature importance but the prior uses the scikit-learn model.feature_importances_ function and the latter uses SHAP (SHapley Additive exPlanations) values. They look quite similar but aren't the same. SHAP values originate from Shapley values, a concept coming from game theory[13]. They add up to the difference between the expected model output and the actual output for a given input. This means that SHAP values provide an accurate[14] and local interpretation of the model's prediction for a given input. Nonetheless, we will just use it as an explainability tool to ease our model interpretation. Figures 23 and 26 just give the absolute value impact of features. Combine them with Figure 25 and we can extract the following insights:

1. Director_popularity has a positive effect on voter ratings. Movies of popular directors do better and vice versa.

2. Surprisingly, longer movies are preferred by the audience.

3. Movies with the genre Drama (could also be a sub-genre e.g. Romance and Drama) did better than movies that weren't Drama.

4. Low-budget films did better than high-budget ones. The impact on both sides wasn't that substantial but important enough.

5. As mentioned in the linear regression section, Actor_3's popularity was more important than the other 2 actors. If Actor_3 is popular that directly implies that at least 3 actors in the film are prominent which positively affects ratings.

6. Some genres like Documentary, Thriller, and Animation did better than others.

7. production_company popularity had an indiscernible effect.

8. Movie plots containing 'tags' like new, york, city, love, life, trouble, comic, evil, country, family, president, etc. again had minute impacts but these features were important enough to make it to the top 20 features among 536.

Although the error hasn't gotten close to what we would hope for, in comparison to other models implemented so far, the improvement looks drastic. The residual analysis plot in Fig 25 can also confirm that out of approximately 4500 movies, the residual for most of the movies is close to zero (except for some obvious outliers).

4.3 XGBoost

Extreme Gradient Boosting (XGBoost) is a highly efficient and effective implementation of the gradient boosting algorithm. It works well with complex datasets to capture the non-linearity. In the hopes of getting better results, we applied this trained model with various combinations (18) of hyperparameters. We used GridSearchCV to apply different permutations of the following parameters:
'learning_rate': [0.1, 0.5, 1.0],
'n_estimators': [10, 50],
'max_depth': [3, 4, 8]

Best Parameters:
learning_rate: 0.1
max_depth: 4
n_estimators: 50
Mean Squared Error (Best Model): ***0.565****673093186695*

R-squared: ***0.554****692648455569*
Mean Absolute Error: ***0.581****897662097064*

These results are very similar to the ones we got from Random Forest. Hence, we explore this just as we did Random Forest and extract any difference in insight.

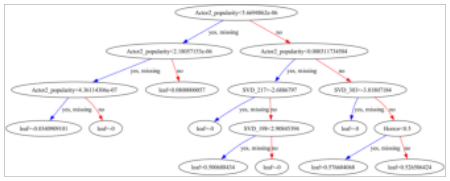

Figure 29: Low-quality image of the first tree plotted by XGBoost

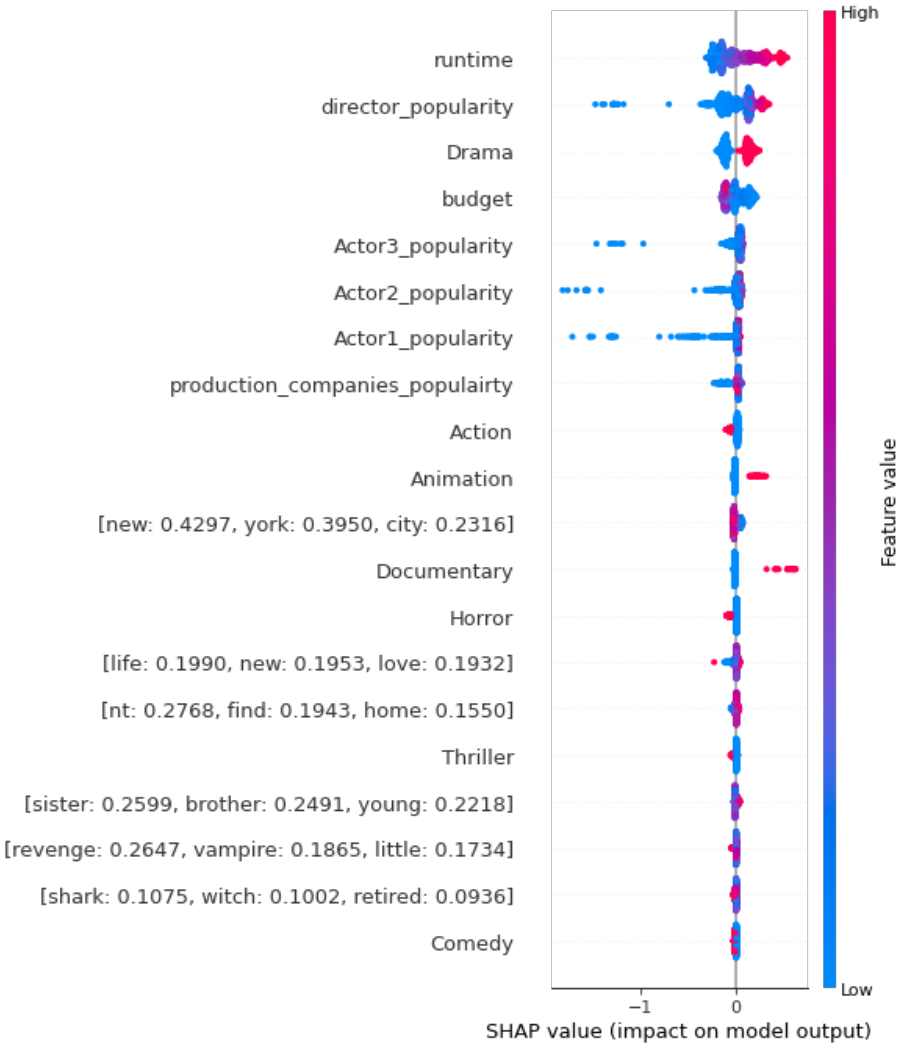

Figure 30: Plot summary with SHAP values

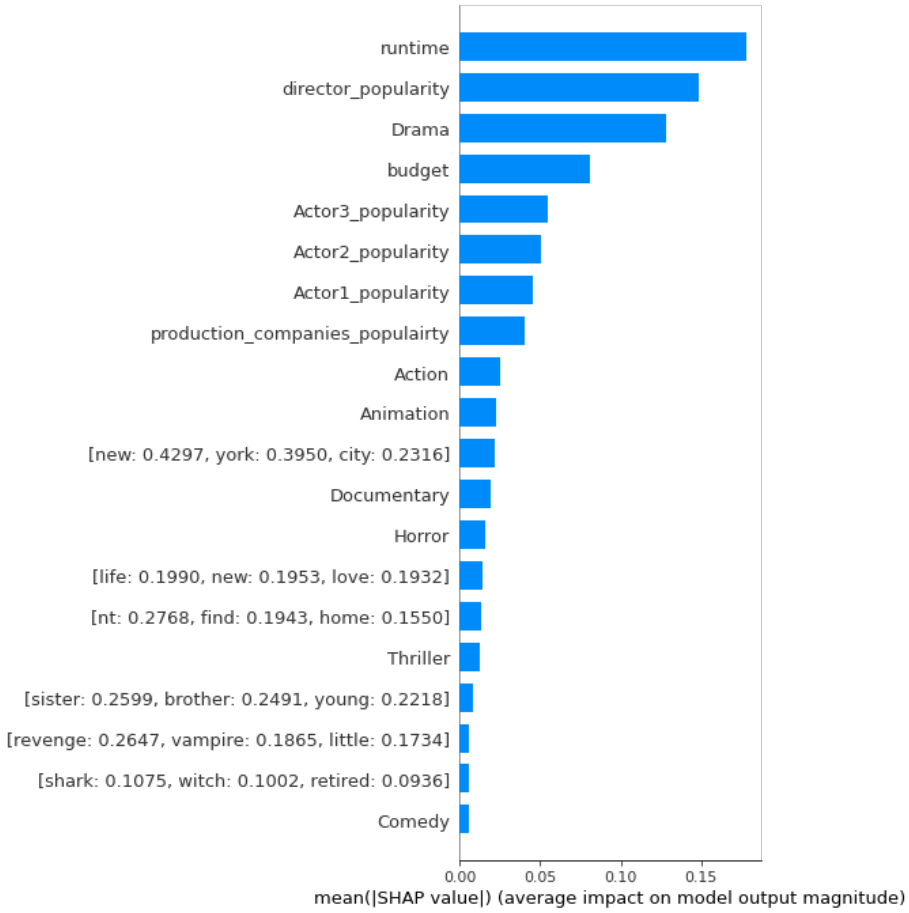

Figure 31: Feature importance with SHAP values

Insights:

1. The runtime and director_popularity switched positions but remain the top 2 significant features.

2. The importance of Actor1_popularity and production_companies popularity has moved up and genres like Action, Documentary, and Animation have shifted but still stay very relevant to the voter ratings.

3. The plot summary 'tags' see some new additions with brother, sister, shark, witch, retired, etc. which means that these tags are also very popular in driving voter ratings (mostly positively).

Although XGBoost performed comparably to Random Forest, we will declare Random Forest the **winner** not just because of the minuscule difference in errors

but also because the computational complexity of Random Forest is lesser than XGBoost, its consistently good performance (in all iterations of our code; not mentioned) shows that it should be chosen as the final model.

5 Experiments

For the sake of coherence and continuity, we have added all empirical results and visualizations in the method, along with the techniques we applied to achieve those results. We extracted many meaningful insights whilst comparing and contrasting model performances. While we took every step in our approach very carefully, we did however train and test Random Forest and XGBoost for a dataset where we applied only Normalization and no Standardization, just for the sake of completeness and satisfying our curiosity. We saw that the results weren't very different. The errors were relatively the same, although the order of certain features in the feature importance plots shifted. Again, that's normal given that it happened within the RF and XGBoost models with the "combination" data. We haven't attached the summary plots for that here but they are available in the code that will be attached.

6 Conclusion

As mentioned before, Random Forest is our choice for the TMDB500 Movie Dataset. It shows an MAE of 0.58 which means our predictions are off by a measure of +/- 0.5 which is a decent accuracy to have. As compared to XGBoost, Random Forest performs well when there are complex interactions or non-linear relationships in the data and when the data doesn't have a clear gradient for boosting to exploit. The industry and dataset in question here, do not demand high accuracies. For an industry like healthcare or Finance, accuracy would be of utmost priority because a skewness of 0.1/0.2 could cause serious repercussions. But for the movie industry, 0.5 is a good ballpark figure. Overall, we've covered a lot of new topics and techniques in this project and had an opportunity to learn and implement a lot of them using the TMDB dataset. To conclude, this project has successfully made inferences while finding the best model for the dataset. The results of this work can be used to allocate appropriate budgets to different aspects of producing a movie.

7 Links

1. The difference of kernels in SVM?
2. XGBoost Mathematics Explained
3. Predict Movie Ratings via Machine Learning
4. Predicting Movie Success..!

5. Normalization, Standardization and Normal Distribution
6. How, When, and Why Should You Normalize / Standardize / Rescale Your Data?
7. When and Why to Standardize Your Data
8. Truncated SVD for Dimensionality Reduction in Sparse Feature Matrices
9. Singular Value Decomposition vs Eigendecomposition for Dimensionality Reduction
10. PCA — What Is Principal Component Analysis and How It Works?
11. Text Classification Using TF-IDF
12. Explainable AI (XAI) with SHAP - regression problem
13. Keyword Extraction Methods from Documents in NLP
14. SHAP (SHapley Additive exPlanations)
15. Feature Engineering: Scaling, Normalization, and Standardization

References

[1] Tmdb 5000 movie dataset. https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata/data.

[2] Predicting imdb ratings of new movies. https://medium.com/web-mining-is688-spring-2021/predicting-imdb-ratings-of-new-movies-2b39459fee9a.

[3] Predict movie ratings via machine learning. https://www.kaggle.com/code/bhsraman/predict-movie-ratings-via-machine-learning.

[4] Rotten tomatoes movie rating prediction. https://www.kdnuggets.com/2023/06/data-science-project-rotten-tomatoes-movie-rating-prediction-first-approach.html.

[5] Tmdb: Popularity trending. https://developer.themoviedb.org/docs/popularity-and-trending.

[6] Tmdb: Production info. https://www.themoviedb.org/bible/movie/59f3b16d9251414f20000008.

[7] Actors clusterization by popularity. https://www.kaggle.com/code/artempanin/actors-clusterization-by-popularity.

[8] Text classification using tf-idf. https://medium.com/swlh/text-classification-using-tf-idf-7404e75565b8.

[9] Singular value decomposition (svd), demystified. https://towardsdatascience.com/singular-value-decomposition-svd-demystified-57fc44b802a0.

[10] Feature engineering: Scaling, normalization, and standardization. https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/#:~:text=A.%20Standardization%20centers%20data%20around,the%20minimum%20and%20maximum%20values.

[11] Know the best evaluation metrics for your regression model ! https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/#:~:text=Commonly%20used%20metrics%20include%20mean,characteristics%20of%20the%20regression%20problem.

[12] The rbf kernel in svm: A complete guide. https://www.pycodemates.com/2022/10/the-rbf-kernel-in-svm-complete-guide.html#:~:text=Unlike%20linear%20or%20polynomial%20kernels,be%20separable%20using%20a%20hyperplane.

[13] Shap values explained exactly how you wished someone explained to you. https://towardsdatascience.com/shap-explained-the-way-i-wish-someone-explained-it-to-me-ab81cc69ef30.

[14] An introduction to shap values and machine learning interpretability. https://www.datacamp.com/tutorial/introduction-to-shap-values-machine-learning-interpretability.