Data Mining

MOVIE RECOMMENDATION SYSTEM USING NEURAL NETWORK BASED HYBRID ENSEMBLE OF VARIOUS FILTERING TECHNIQUES AND SCORE PREDICTION

Team Members :
Romit Mondal - 18075051
Shayak Das - 18075056
Archit Saha - 18075010
Radhika Singh - 18075046

Project Sequence

Obtaining Dataset

Data Cleaning

Data Transformation

Data PreProcessing

Data Visualization

Building Score Predictor

Building Recommender Models

Studying Accuracy

Overview

We will use **TMDB 5000 Movies Dataset** from Kaggle throughout our Project.

When new movies are produced, every stakeholder is interested in the financial success of the intended film. To predict the success of the movie, specific methods are applied, such as market investigations or analysis, which cost a lot. This proposed work aims to develop a model based upon the data mining techniques that may help in predicting the success or failure of a movie in advance, thereby reducing the specific level of uncertainty. The decision-maker (movie creators and stakeholders) has all the information about the exact outcome of the decision before he or she makes a choice (release of the movie).

Data is being used to create more efficient systems, and this is where Recommendation Systems come into play. Recommendation Systems are a type of **information filtering systems** as they increase the quality of search results and provide details that are more relevant to the search item.


Data Preprocessing
NETFLIX
STRANGER THINGS
DAREDEVIL
LUKE CAGE
THE RANCH
SKYLANDERS ACADEMY
THE DO OVER
THE 100
ROMAN EMPIRE
BROOKLYN NINE-NINE
MARCO POLO
LOVE
AUDREY & DAISY
BOJACK HORSEMAN
SUITS
THE CROWN
ELEMENTARY
Little Baby Bum
FULLER HOUSE
OGGY AND THE COCKROACHES
HEAD 2 HEAD
RICK AND MORTY
Larva
FREQUENCY
Tales by Light

Data Pre Processing

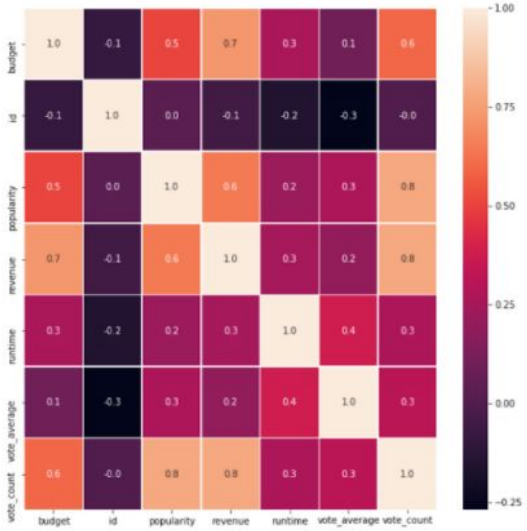
1. Data Cleaning: The data can have many unnecessary and missing components. To handle this, data cleaning is done. It involves handling missing data, noisy data etc.

2. Data Transformation

3. Dimensionality Reduction: This reduces the size of data by encoding mechanisms. It can either be lossy or lossless. If after reconstruction from compressed data, original data can be retrieved, such reduction is named lossless reduction else it is called lossy reduction. The two effective methods of dimensionality reduction are: Wavelet transforms and PCA (Principal Component Analysis).

To achieve Data Pre-Processing, we will use Data Visualization and **Exploratory Data Analysis** on the given Dataset.

```
In [7]:
# correlation map
f,ax = plt.subplots(figsize = (10,10))
sns.heatmap(data.corr(), annot = True, linewidths=.5, fmt = '.1f', ax = ax)
plt.show()
```



That includes eliminating highly linearly dependent attributes or redundant attributes by checking the correlation matrix.

Important Selected Attributes:

- **Overview**: Description of the movie can suggest the type of movie and give us the keywords in a movie for example say we have Superman, the movie is based on a comic so most likely comic will be a keyword in the overview.
- **Genres**: Genre is very indicative of the taste of the user and thus must be used.
- **Keywords**: These are keywords pre existing in the data that can be said to be tags representing a movie.
- **Actors**: A person watches a movie if he likes a particular actor and that is why we need to include main actors but not all actors as supporting actors are not much of interest.
- **Director**: People often go for watching a movie based on the director as a director who has a good reputation in the industry is more sort after.
- **Vote_average:** The score of the movie and how much it has been rated overall.
- **Budget:** The budget involved in the production of the movies.
- **Revenue:** The total revenue generated from the movie as a return.
- **Popularity:** These are the scores based on how popular a movie has been.

Cleaning

Keywords will play an important role in the functioning of the engine. Indeed, recommendations will be based on similarity between films and to gauge such similarities, I will look for films described by the same keywords. Hence, the content of the **plot_keywords** variable deserves some attention since it will be extensively used.

Grouping by roots :

I collect the keywords that appear in the plot_keywords variable. This list is then cleaned using the NLTK package. Finally, I look for the number of occurence of the various keywords.

```
1 {'alien', 'alienation'} 2
2 {'spy', 'spying'} 2
3 {'vigilantism', 'vigilante'} 2
4 {'terror', 'terrorism'} 2
5 {'flood', 'flooding'} 2
6 {'spiders', 'spider'} 2
7 {'horses', 'horse'} 2
8 {'musical', 'music'} 2
9 {'animal', 'animation', 'anime'} 3
10 {'compass', 'compassion'} 2
11 {'train', 'training'} 2
12 {'sail', 'sailing'} 2
13 {'time traveler', 'time travel'} 2
14 {'orcs', 'orc'} 2
```

We are replacing words by their main forms , keywords by their most frequent occurrence .

Cleaning

Groups of *synonyms :*

I clean the list of keywords in two steps. As a first step, I suppress the keywords that appear less than 5 times and replace them by a synonym of higher frequency. As a second step, I suppress all the keywords that appear in less than 3 films .

```
"outlander                  " in keywords list -> False 0
"extraterrestrial           " in keywords list -> True 4
"extraterrestrial being     " in keywords list -> False 0
"foreigner                  " in keywords list -> False 0
"stranger                   " in keywords list -> True 7
"noncitizen                 " in keywords list -> False 0
"alien                      " in keywords list -> True 80
"unknown                    " in keywords list -> False 0
```

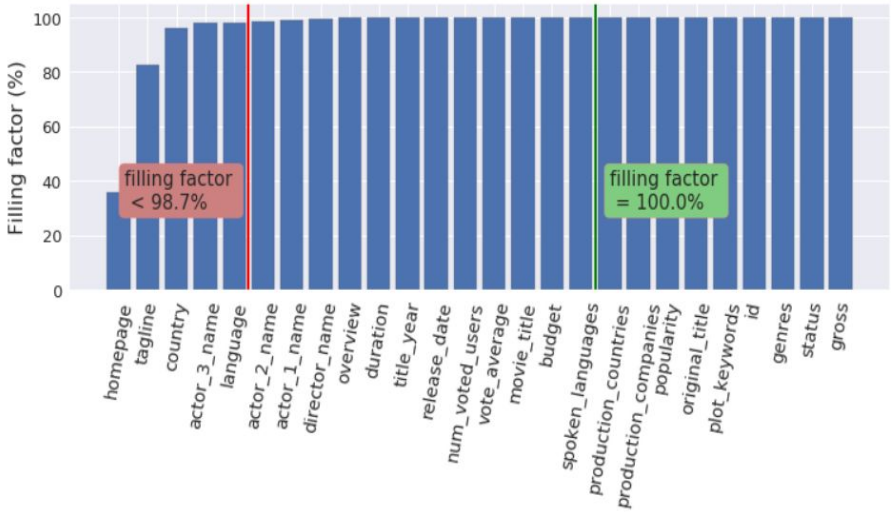
Replacement of Similar Keywords :

```
narcism      -> narcissism   (init: [('narcissism', 1), ('narcism', 1)])
apparition   -> shadow       (init: [('shadow', 3), ('phantom', 3), ('apparition', 1)])
macao        -> macau        (init: [('macau', 1), ('macao', 1)])
regent       -> trustee      (init: [('trustee', 1), ('regent', 1)])
civilization -> culture      (init: [('culture', 2), ('civilization', 1)])
ark          -> ark of the covenant (init: [('ark of the covenant', 2), ('ark', 1)])
automaton    -> zombie       (init: [('zombie', 45), ('robot', 27), ('automaton', 1)])
------------------------------------------------------------------------------------
---
The replacement concerns 5.96% of the keywords.
```

Handling Missing Values

Handling Missing Keywords : I try to fill missing values in the **plot_keywords** variable using the words of the title. To do so, I create the list of synonyms of all the words contained in the title and I check if any of these synonyms are already in the keyword list. When it is the case, I add this keyword to the entry.

Handling Other Missing Values : I had a look at the correlation between variables and found that a few of them showed some degree of correlation, with a Pearson's coefficient > 0.5. I will use this finding to fill the missing values of the **gross**, **num_critic_for_reviews**, **num_voted_users** , and **num_user_for_reviews** variables. To do so, I will make regressions on pairs of correlated variables. First, I define a function that impute the missing value from a linear fit of the data. This function takes the dataframe as input, as well as the names of two columns. A linear fit is performed between those two columns which is used to fill the holes in the first column that was given.



After Missing Values are taken care of , the plot of Filling Factor vs Attributes takes the following shape.

JSON to String

The next job is to convert the data in the before mentioned columns from json to string format. This step of preprocessing is very essential in order to extract the useful information from the data.

```
# changing the genres column from json to string
movies['genres'] = movies['genres'].apply(json.loads)
for index,i in zip(movies.index,movies['genres']):
    list1 = []
    for j in range(len(i)):
        list1.append((i[j]['name'])) # the key 'name' contains the name of the genre
    movies.loc[index,'genres'] = str(list1)

# changing the keywords column from json to string
movies['keywords'] = movies['keywords'].apply(json.loads)
for index,i in zip(movies.index,movies['keywords']):
    list1 = []
    for j in range(len(i)):
        list1.append((i[j]['name']))
    movies.loc[index,'keywords'] = str(list1)

# changing the production_companies column from json to string
movies['production_companies'] = movies['production_companies'].apply(json.loads)
for index,i in zip(movies.index,movies['production_companies']):
    list1 = []
    for j in range(len(i)):
        list1.append((i[j]['name']))
    movies.loc[index,'production_companies'] = str(list1)

# changing the cast column from json to string
credits['cast'] = credits['cast'].apply(json.loads)
for index,i in zip(credits.index,credits['cast']):
    list1 = []
    for j in range(len(i)):
        list1.append((i[j]['name']))
    credits.loc[index,'cast'] = str(list1)
```

One Hot Encoding

One hot encoding is a process by which categorical variables are converted into a form that could be provided to data processing paradigms to do a better job.
For example, the name of directors in string format can't be given as an input since this doesn't make much sense. But we can convert each director's name to an integer value and convert that to a binary array which tells which one is the director.

```
0          James Cameron
1        Gore Verbinski
2            Sam Mendes
3     Christopher Nolan
4        Andrew Stanton
Name: director, dtype: object
```

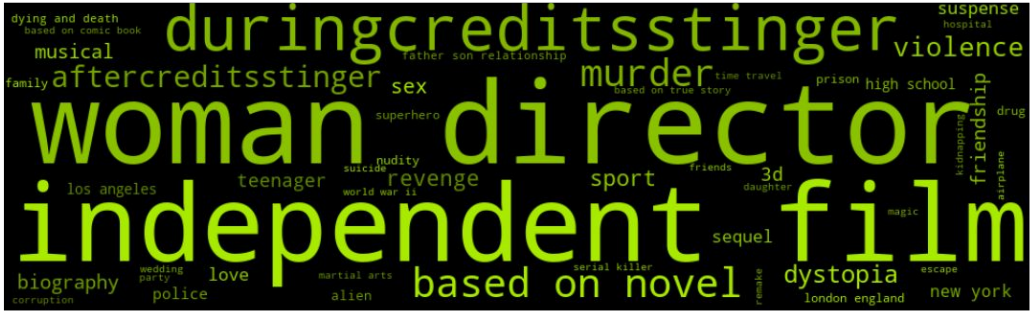
	original_title	genres	vote_average	genres_bin	cast_bin	new_id	director	director_bin	words_bin	popularity	budget	revenue
0	Avatar	[Action, Adventure, Fantasy, ScienceFiction]	7.2	[1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	0	James Cameron	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	150.437577	237000000	2787965087
1	Pirates of the Caribbean: At World's End	[Action, Adventure, Fantasy]	6.9	[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, ...	1	Gore Verbinski	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	139.082615	300000000	961000000
2	Spectre	[Action, Adventure, Crime]	6.3	[1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	2	Sam Mendes	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	107.376788	245000000	880674609
3	The Dark Knight Rises	[Action, Crime, Drama, Thriller]	7.6	[1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, ...	3	Christopher Nolan	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	112.312950	250000000	1084939099
4	John Carter	[Action, Adventure, ScienceFiction]	6.1	[1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, ...	4	Andrew Stanton	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	43.926995	260000000	284139100

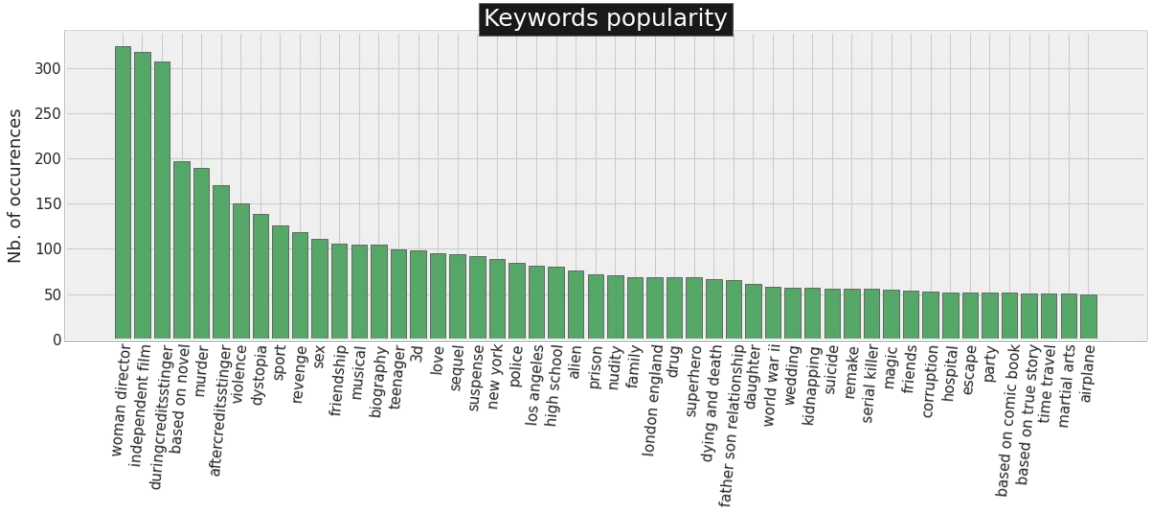

Data Visualization

Keywords Popularity

This visualization helps us understand the distribution of the count of keyword present in the overview attribute.

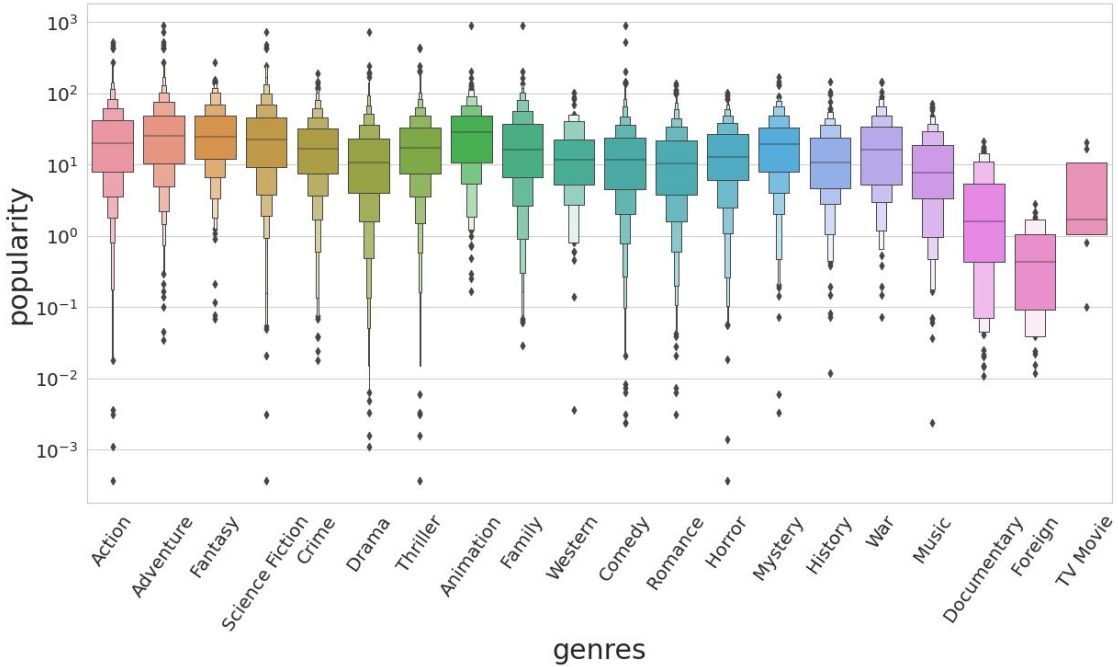
Using Wordcount library, we can see the visualization better where keywords with more occurrence has bigger font.





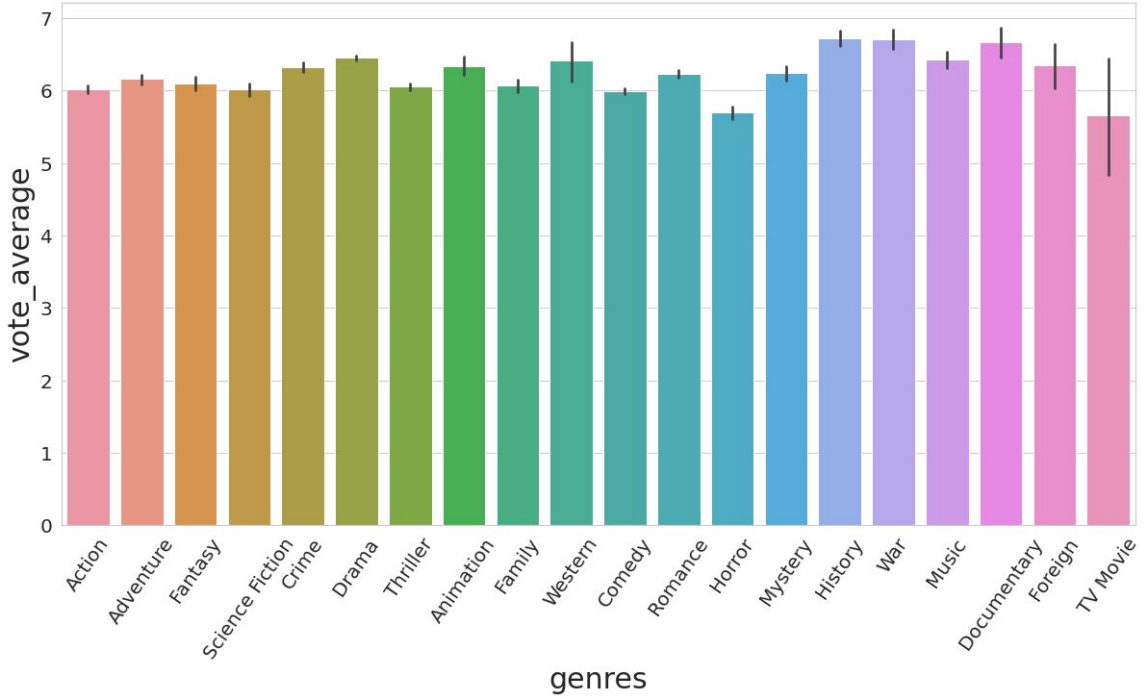
Popularity vs Genre

This visualization helps us understand the popularity distribution of each genre using the 5 Number Summary of Box Plot distribution.



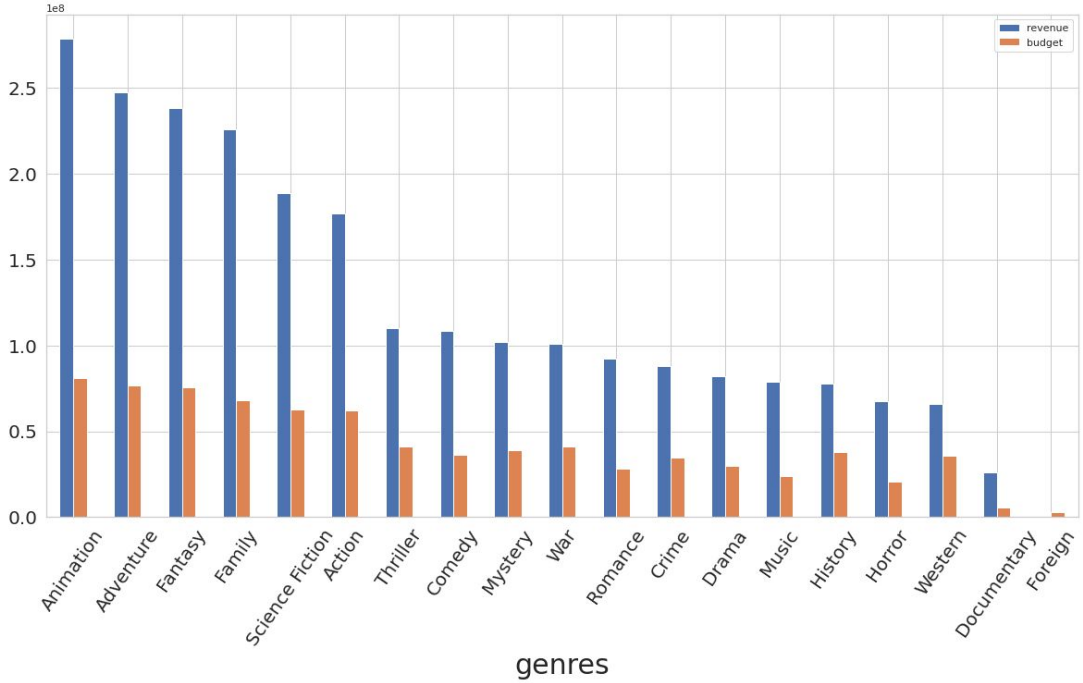
Vote Average vs Genre

This visualization helps us understand the histogram of vote average with respect to genre.



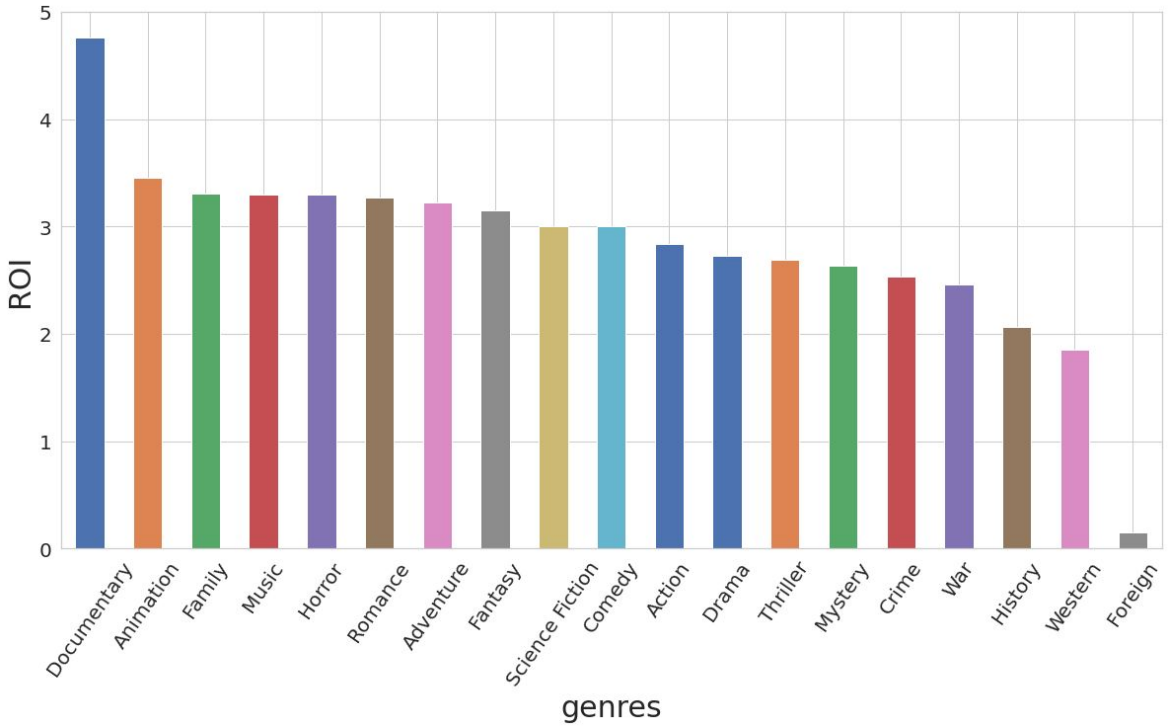
Revenue/Budget

This visualization helps us understand the relation of the budget of the movie and revenue generated by movies of different genres.



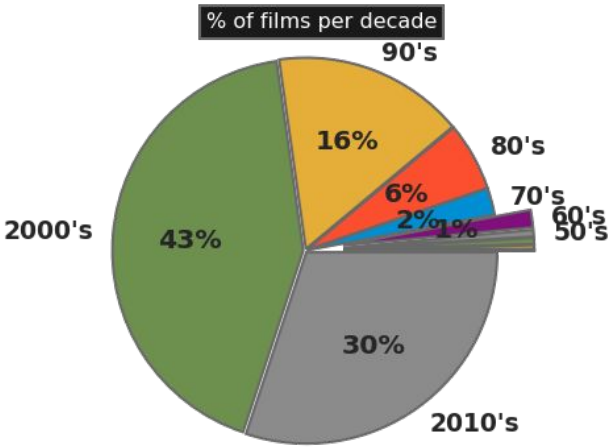
Return on Investment

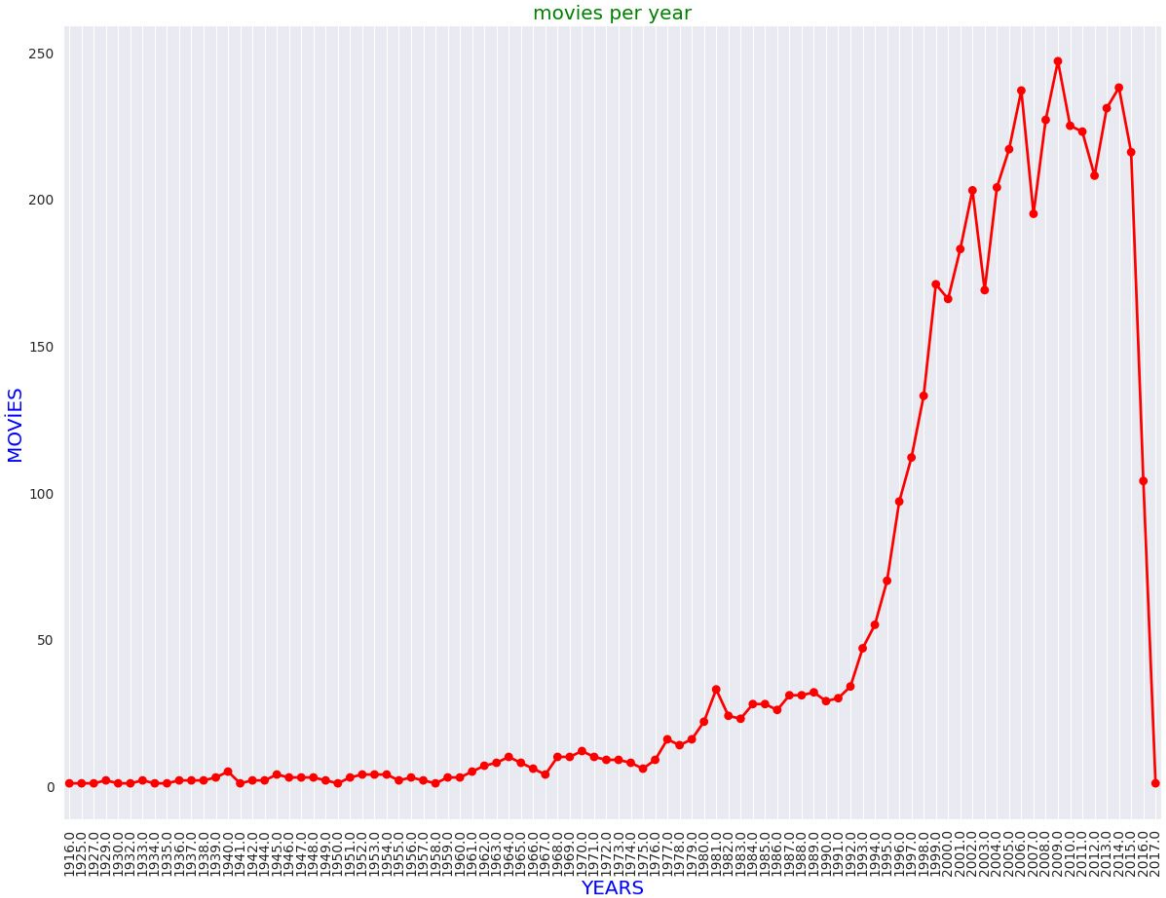
Since budget and revenue does not clearly tell how much revenue is generated by the budget. Thus we plot the ratio of revenue generated with respect to the budget which helps us better understand the return on investment.



Movies per year

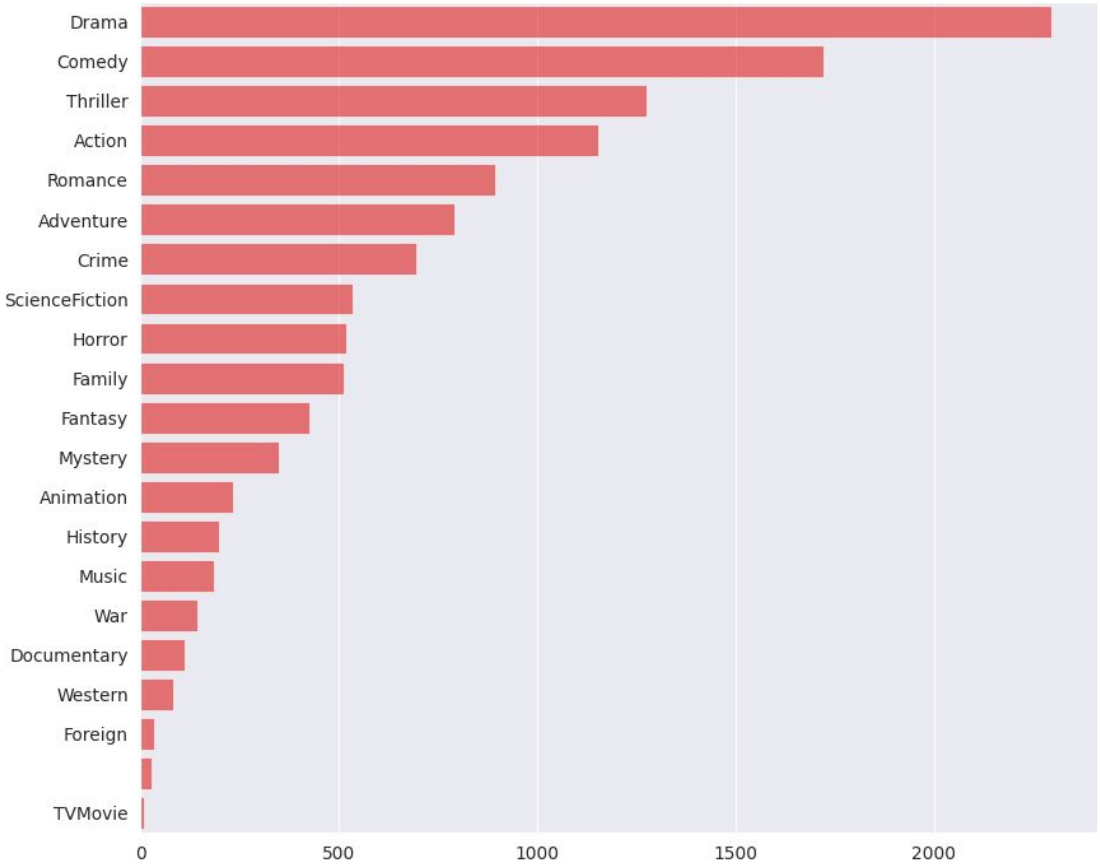
This visualization helps us understand the distribution of the number of movies per year and per decade and how the trend of movie production has changed gradually.





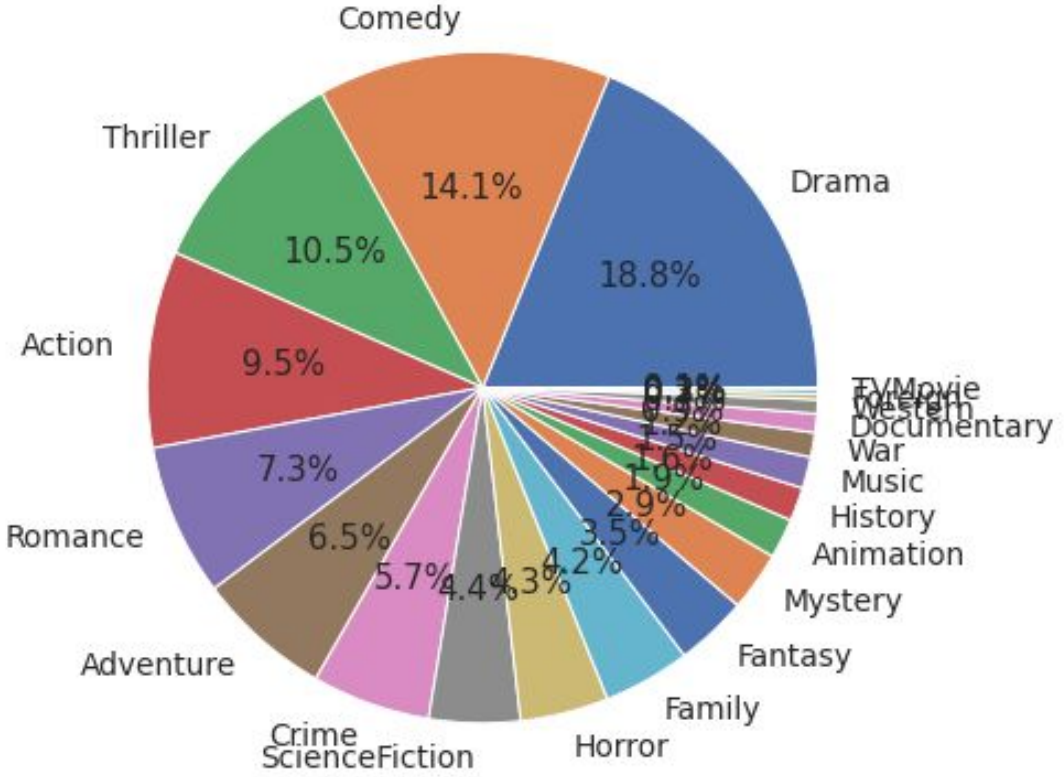
Movies per genre

This visualization helps us understand the distribution of the number of movies per genre and how this data have a major role in score prediction and movie recommendation.



Movies per genre

This pie chart visualization helps us understand the the previous data in a better format, which shows the percentage of movies per genre in a better format.

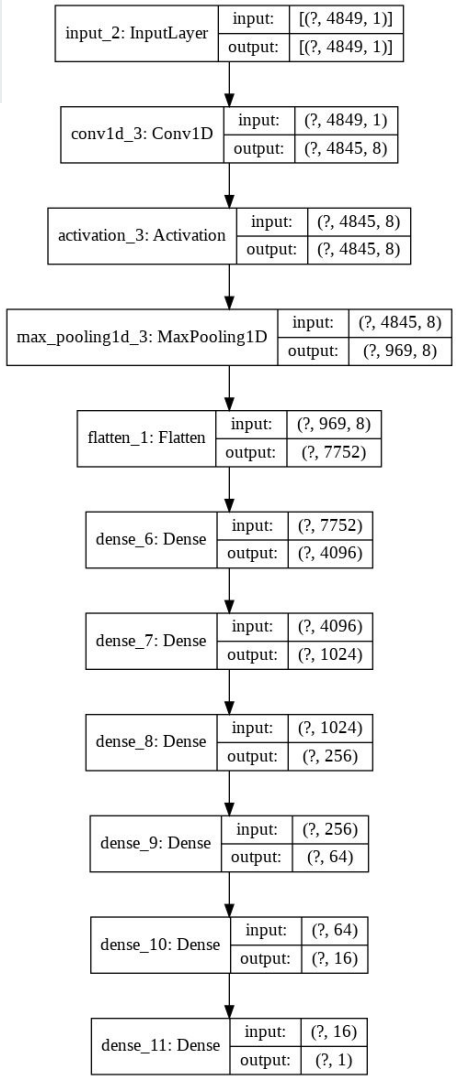



Score Prediction

Hybrid Neural Network for prediction of Movie Score

We build a 1-D Neural Network which takes the data of genres, cast, director, keywords, popularity, budget and revenue as input and is trained to predict the vote average score.
The feed-forward Neural Network is built using a hybrid of CNN and MLP using Convolution, ReLU Activation, MaxPool and Dense layers.
We used mean squared logarithmic error as the loss function and cosine proximity as the evaluation metric.

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 4849, 1)]         0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 4845, 8)           48
_________________________________________________________________
activation_3 (Activation)    (None, 4845, 8)           0
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 969, 8)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 7752)              0
_________________________________________________________________
dense_6 (Dense)              (None, 4096)              31756288
_________________________________________________________________
dense_7 (Dense)              (None, 1024)              4195328
_________________________________________________________________
dense_8 (Dense)              (None, 256)               262400
_________________________________________________________________
dense_9 (Dense)              (None, 64)                16448
_________________________________________________________________
dense_10 (Dense)             (None, 16)                1040
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 17
=================================================================
Total params: 36,231,569
Trainable params: 36,231,569
Non-trainable params: 0
```



Cosine Similarity Scores

We can now compute a similarity score. There are several candidates for this; such as the euclidean, the Pearson and the cosine similarity scores. There is no right answer to which score is the best. Different scores work well in different scenarios and it is often a good idea to experiment with different metrics.

We will be using the cosine similarity to calculate a numeric quantity that denotes the similarity between two movies. We use the cosine similarity score since it is independent of magnitude and is relatively easy and fast to calculate. Mathematically, it is defined as follows:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$


Movie Recommendation

Using KNN for Movie Recommendation

We use K Nearest Neighbours for recommending movies based on similarity scores, based on the **Cosine Similarity Scores,** of the selected movie with all other movies in the dataset. The K closest movies based on the selected features are shown as recommended movies.

In the given example, K is 10 and the movie selected is The Godfather: Part III.

```
predict_score('Godfather', 10)
```

```
Selected Movie:  The Godfather: Part III

Recommended Movies:

The Rainmaker | Genres: 'Crime','Drama','Thriller' | Rating: 6.7
The Godfather | Genres: 'Crime','Drama' | Rating: 8.4
The Cotton Club | Genres: 'Crime','Drama','Music','Romance' | Rating: 6.6
Apocalypse Now | Genres: 'Drama','War' | Rating: 8.0
New York Stories | Genres: 'Comedy','Drama','Romance' | Rating: 6.2
Peggy Sue Got Married | Genres: 'Comedy','Drama','Fantasy','Romance' | Rating: 5.9
End of Watch | Genres: 'Crime','Drama','Thriller' | Rating: 7.2
Only God Forgives | Genres: 'Crime','Drama','Thriller' | Rating: 5.6
Hannibal Rising | Genres: 'Crime','Drama','Thriller' | Rating: 6.0
Savages | Genres: 'Crime','Drama','Thriller' | Rating: 6.2


The predicted rating for The Godfather: Part III is: 6.680000
The cumulative mean rating for The Godfather: Part III is: 6.866452
The learned rating for The Godfather: Part III is: 6.226976
The actual rating for The Godfather: Part III is 7.100000
```

Movie Recommendation Function

- Get the index of the movie given its title.
- Get the list of cosine similarity scores for that particular movie with all movies. Convert it into a list of tuples where the first element is its position and the second is the similarity score.
- Sort the aforementioned list of tuples based on the similarity scores; that is, the second element.
- Get the top K elements of this list. Ignore the first element as it refers to self (the movie most similar to a particular movie is the movie itself).
- Return the titles corresponding to the indices of the top elements.

Predicted Scores

As we can see in the previous slide, we predict the scores of the given movie along with the movie recommendations, which turns out to be the measure to see how accurate the model works.

- **Actual Rating:** This is the actual rating which is the data given in “vote_average” column. This is the base rating, which is used as a measure to understand the accuracy of movie recommendation.
- **Average Rating:** This is the predicted rating which we get by simply calculating the average of the rating of the recommended movies, which turns out to be quite accurate due to cosine similarity of movies.
- **Cumulative Mean Rating:** This is the predicted rating which we get by calculating the cumulative mean of the rating of the recommended movies with the inverse of distance calculated using cosine similarity scores.
- **Artificially Learned Rating:** This is the predicted rating which we get from the Hybrid Neural Network rating predictor, which also succeeds in predicting the score quite accurately.

```
The predicted rating for The Godfather: Part III is: 6.680000
The cumulative mean rating for The Godfather: Part III is: 6.866452
The learned rating for The Godfather: Part III is: 6.226976
The actual rating for The Godfather: Part III is 7.100000
```

Content-Based Recommender

In this recommender system the content of the movie (cast, crew, keyword, tagline etc) is used to find its similarity with other movies. Then the movies that are most likely to be similar are recommended. Some of the main functions we have added in the code:

- **Function collecting some variables content**: the entry_variables() function returns the values taken by the variables 'director_name', 'actor_N_name' (N ∈ [1:3]) and 'plot_keywords' for the film selected by the user.
- **Function adding variables to the dataframe**: the function *add_variables()* add a list of variables to the dataframe given in input and initialize these variables at 0 or 1 depending on the correspondence with the description of the films and the content of the REF_VAR variable given in input.
- **Function creating a list of films**: the *recommand()* function create a list of N (= 31) films similar to the film selected by the user.
- **Function comparing 2 film titles**: the sequel *sequel()* function compares the 2 titles passed in input and defines if these titles are similar or not.
- **Function giving marks to films**: the *criteria_selection()* function gives a mark to a film depending on its IMDB score, the title year and the number of users who have voted for this film.
- **Function adding films**: the *add_to_selection()* function complete the *film_selection* list which contains 5 films that will be recommended to the user. The films are selected from the *parametres_films* list and are taken into account only if the title is different enough from other film titles.
- **Function filtering sequels**: the *remove_sequels()* function remove sequels from the list if more that two films from a serie are present. The older one is kept.
- **Main function**: create a list of 5 films that will be recommended to the user.

Content-Based Recommender

While building the recommendation engine, we are quickly faced to a big issue: the existence of sequels make that some recommendations may seem quite dumb ... As an example, somebody who enjoyed *"Pirates of the Caribbean: Dead Man's Chest"* would probably not like to be advised to watch its previous sequels.

Hence, I tried to find a way to prevent that kind of behaviour and I concluded that the quickest way to do it would be to work on the film's titles. To do so, I used the **fuzzywuzzy** package to build the *remove_sequels()* function. This function defines the degree of similarity of two film titles and if too close, the most recent film is removed from the list of recommendations.

Below is the output when del_sequels is turned off and on .

```
dum = find_similarities(df, 12, del_sequels = False, verbose = True)
```

```
-------------------------------------------------------------------------------
---
QUERY: films similar to id=12 -> 'Pirates of the Caribbean: Dead Man's Chest'
nº1      -> Pirates of the Caribbean: Dead Man's Chest
nº2      -> Pirates of the Caribbean: At World's End
nº3      -> Pirates of the Caribbean: The Curse of the Black Pearl
nº4      -> Pirates of the Caribbean: On Stranger Tides
nº5      -> Cutthroat Island
```

```
dum = find_similarities(df, 12, del_sequels = True, verbose = True)
```

```
-------------------------------------------------------------------------------
---
QUERY: films similar to id=12 -> 'Pirates of the Caribbean: Dead Man's Chest'
nº1      -> Pirates of the Caribbean: The Curse of the Black Pearl
nº2      -> Cutthroat Island
nº3      -> The Hobbit: An Unexpected Journey
nº4      -> The 13th Warrior
nº5      -> Red Sonja
```

Content-Based Recommender (Text Mining)

We will compute pairwise similarity scores for all movies based on their plot descriptions and recommend movies based on that similarity score. The plot description is given in the **overview** feature of our dataset.

Inverse Document Frequency is the relative count of documents containing the term is given as **log(number of documents/documents with term)** The overall importance of each word to the documents in which they appear is equal to **TF * IDF .** This will give you a matrix where each column represents a word in the overview vocabulary (all the words that appear in at least one document) and each row represents a movie, as before.This is done to reduce the importance of words that occur frequently in plot overviews and therefore, their significance in computing the final similarity score.

The motivation behind IDF(inverse document frequency) is that all the words in a document are not necessarily useful in modelling the topics for that document.

Formula used for calculating the IDF is :- np.log10(number of documents/number of documents containing that word)

Since we have used the TF-IDF vectorizer, calculating the dot product will directly give us the cosine similarity score. Therefore, we will use sklearn's **linear_kernel()** instead of cosine_similarities() since it is faster.

Content-Based Recommender (Text Mining)

On calling the function with a particular movie name , it gives the following output : >>

```
In [16]: get_recommendations('The Dark Knight Rises')
```

```
Out[16]:
65                              The Dark Knight
299                              Batman Forever
428                              Batman Returns
1359                                    Batman
3854    Batman: The Dark Knight Returns, Part 2
119                               Batman Begins
2507                                  Slow Burn
9           Batman v Superman: Dawn of Justice
1181                                      JFK
210                              Batman & Robin
Name: title, dtype: object
```

While our system has done a decent job of finding movies with similar plot descriptions, the quality of recommendations is not that great. "The Dark Knight Rises" returns all Batman movies while it is more likely that the people who liked that movie are more inclined to enjoy other Christopher Nolan movies. This is something that cannot be captured by the present system.

Collaborative Filtering

Our content based engine suffers from some severe limitations. It is only capable of suggesting movies which are close to a certain movie. That is, it is not capable of capturing tastes and providing recommendations across genres.

Also, the engine that we built is not really personal in that it doesn't capture the personal tastes and biases of a user. Anyone querying our engine for recommendations based on a movie will receive the same recommendations for that movie, regardless of who she/he is.

Therefore, in this section, we will use a technique called Collaborative Filtering to make recommendations to Movie Watchers. It is basically of two types:-

1. User-Based Filtering : These systems recommend products to a user that similar users have liked. For measuring the similarity between two users we can either use pearson correlation or cosine similarity.

   Although computing user-based CF is very simple, it suffers from several problems. One main issue is that users' preference can change over time. It indicates that precomputing the matrix based on their neighboring users may lead to bad performance. To tackle this problem, we can apply item-based CF.

Collaborative Filtering

2. **Item-Based Filtering** : Instead of measuring the similarity between users, the item-based CF recommends items based on their similarity with the items that the target user rated. Likewise, the similarity can be computed with Pearson Correlation or Cosine Similarity. The major difference is that, with item-based collaborative filtering, we fill in the blank vertically, as oppose to the horizontal manner that user-based CF does. The following table shows how to do so for the movie Me Before You :

It successfully avoids the problem posed by dynamic user preference as item-based CF is more static. However, several problems remain for this method. First, the main issue is ***scalability***. The computation grows with both the customer and the product. The worst case complexity is O(mn) with m users and n items. In addition, ***sparsity*** is another concern. Take a look at the above table again. Although there is only one user that rated both Matrix and Titanic rated, the similarity between them is 1. In extreme cases, we can have millions of users and the similarity between two fairly different movies could be very high simply because they have similar rank for the only user who ranked them both.

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You
A	2		2	4	5	2.94*
B	5		4			1
C			5		2	2.48*
D		1		5		4
E			4			2
F	4	5		1		1.12*
Similarity	-1	-1	0.86	1	1	

Collaborative Filtering

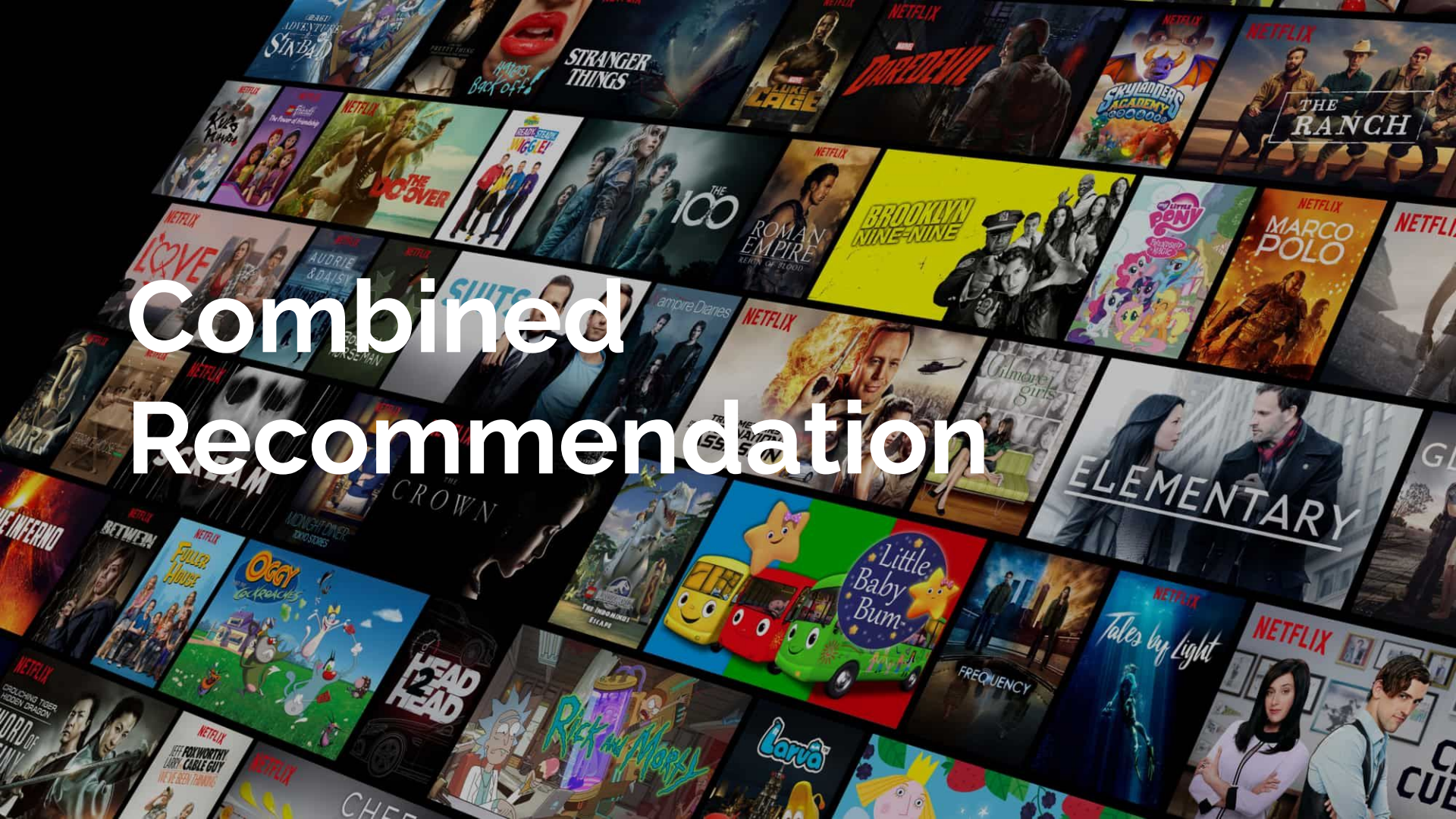
Single Value Decomposition : One way to handle the scalability and sparsity issue created by CF is to leverage a **latent factor model** to capture the similarity between users and items. Essentially, we want to turn the recommendation problem into an optimization problem. We can view it as how good we are in predicting the rating for items given a user. One common metric is Root Mean Square Error (RMSE). **The lower the RMSE, the better the performance**.

Since the dataset we used before did not have userId(which is necessary for collaborative filtering) let's load another dataset. We'll be using the **Surprise** library to implement SVD.

```
In [36]: svd.predict(1, 302, 3)

Out[36]: Prediction(uid=1, iid=302, r_ui=3, est=2.756291780279027, details={'was_impossible': Fa
         lse})
```

For userId 1 and movie with ID 302, we get an estimated prediction of **2.756**. One startling feature of this recommender system is that it doesn't care what the movie is (or what it contains). It works purely on the basis of an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie.

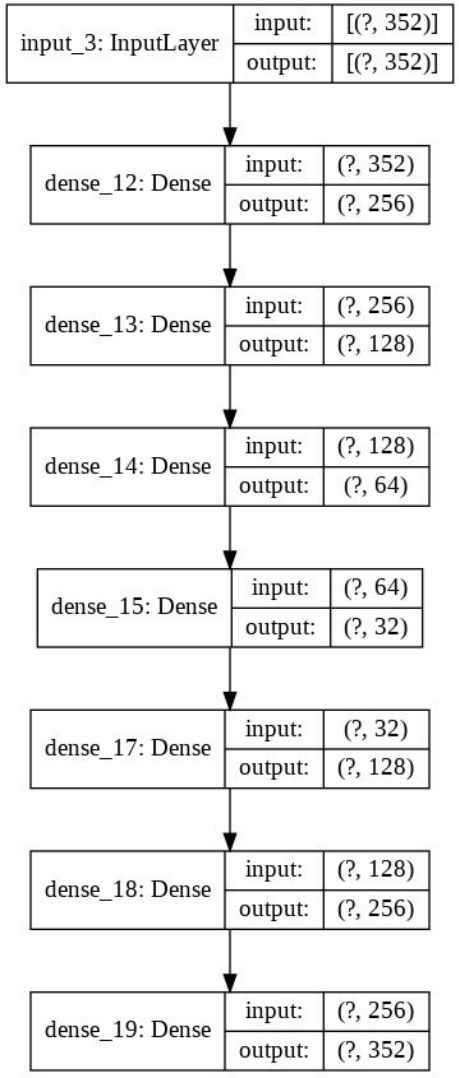

Combined
Recommendation

Projecting Data into Lower Dimension

- Since the data of movies is in plain language we need some way to decompose it into some numerical values and project it to a lower dimension so that we get some meaningful data for our model.
- We use **TFIDF Vectorizer** to convert text data (information of overview, genre, director, etc.) into numerical data which represents how relevant each word in the document is.
- Some more numerical information regarding vote average, budget, revenue, etc. were added.
- The data was **normalized** using sklearn preprocessing.
- **Word embeddings** of each feature were generated to provide a dense representation of words and their relative meanings, with the help of Keras Embedding layers.
- Now we get a set of meaningful normalized data to be used for our ensemble purpose.

Autoencoder model

The autoencoder consists of densely connected layers where the size of layers decrease gradually to the point where we get the encoded data representation projected onto a lower dimension. Then the data is decompressed again with a similar sequence of **dense layers with ELU activation**. The model is compiled with **Mean Squared Error loss** and **Adam optimizer**. The model learns to regenerate the features of movies quite accurately and we use the encoded information to make a hybrid ensemble which gives us the combined movie recommendation from the different models discussed above.



Ensemble using Encoded Data

- We need the set of movies predicted by each of the models and the name of the movie for which we are generating the recommendations.
- The encoded data vector of each movie is taken.
- Cosine similarity score is used as a distance metric to compute the distance of the recommended movies from the original given movie.
- These are convert it into a list of tuples where the first element is the index and the second is the similarity score.
- The aforementioned list of tuples is sorted based on the similarity scores; that is, the second element.
- Get the top K elements of this list.
- We thus get the titles of the movies based on the final combined movie recommendation.

Final Recommended Movies

Selected Movie: The Godfather: Part III

Recommended Movies:

The Rainmaker
Peggy Sue Got Married
Collateral Damage
Absolute Power
City By The Sea
Apocalypse Now
London Has Fallen
We Own the Night
The Interpreter
Only God Forgives

Selected Movie: Avengers: Age of Ultron

Recommended Movies:

Cradle 2 the Grave
Captain America: The Winter Soldier
The Man from U.N.C.L.E.
Ant-Man
Serenity
The Avengers
Megaforce
Unstoppable
Knight and Day
Iron Man 2

Selected Movie: Pulp Fiction

Recommended Movies:

Reservoir Dogs
Kill List
Sin City: A Dame to Kill For
The Golden Compass
The Town
Sliding Doors
Kill Bill: Vol. 1
Cape Fear
Jackie Brown
The Sting

Selected Movie: Pirates of the Caribbean: At World's End

Recommended Movies:

Cutthroat Island
90 Minutes in Heaven
The Scorpion King
Just Like Heaven
Spider-Man 2
The Descendants
Pirates of the Caribbean: Dead Man's Chest
The Mexican
Dungeons & Dragons: Wrath of the Dragon God
Pirates of the Caribbean: The Curse of the Black Pearl

Thank you.