greatlearning

Movie Recommender System using Hybrid filtering and KNN

Batch details	**DSE Online May 2022**
Team members	• Abhinav Anand • Deekshitha AG • Mahathevan S • Lakshmi Priya A • Priyanka S
Domain of Project	Recommender System – Movie Dataset
Proposed project title	Movie Recommender System using Hybrid filtering and KNN
Group Number	Group 7
Team Leader	Abhinav Anand
Mentor Name	Dr. Debasish Roy

Table of Contents

1. Summary of problem statement, data and findings

1.1 Business Objective / Understanding

A movie recommender system is basically a tool that helps streaming media platforms recommend users' favourite movies on the basis of their interests and behaviour.

OTT platforms like Netfix, Amazon Prime Video, Rakuten Viki ,Disney+Hotstar ,etc will recommend movies based on our watchlist. It creates a list of favourite movies

according to each individual profile. Using an AI-based algorithm that analyses the data, it goes through various possible options, and creates a customized list of items that are interesting and relevant to an individual.
Web-based movie recommender systems like MovieLens , Flickmetrix, Popcornic,etc will recommend movies based on our customized answers. Businesses just need to upload the data set and extract inputs from their users. After generating the recommendations out of inputs, businesses can apply preferred filters to adjust the final recommendations.

1.2 Literature Survey - Publications, Application, past and undergoing research

MOVREC is a movie recommendation system designed by D.K. Yadav based on collaborative filtering. Collaborative filtering mainly uses the information that is provided by the user. That information is considered, and movies are recommended to the users which are ordered with highest rating first.

Luis M Capos has proposed a new system which is a combination of Bayesian network and collaborative filtering. The proposed system is optimized for the given problem and provides probability distributions to make useful inferences.

A hybrid system has been proposed by Harpreet Kaur that uses a mix of content as well as collaborative filtering algorithm. Here the context of the movies is also considered while recommending.

The user specific information or item specific information is clubbed to form a cluster by Utkarsh Gupta using chameleon. To predict the rating of an item voting system is used. The proposed system has lower error and has better clustering of similar items.

Urszula Kużelewska proposed clustering as a way to deal with recommender systems. Centroid-based solution and memory-based collaborative filtering methods were used as a basis for comparing effectiveness of the proposed two methods. The result was a significant increase in the accuracy of the generated recommendations when compared to just centroid-based method.

To predict the difficulty level of each case for each trainee Hongli LIn proposed a method called content-boosted collaborative filtering (CBCF).The algorithm is divided into two stages, First the content-based filtering that improves the existing trainee case ratings data and the second being collaborative filtering that provides the final predictions. The CBCF algorithm involves the advantages of both CBF and CF, while at the same time, overcoming both their disadvantages.

1.3 Relevant Papers

- https://www.researchgate.net/publication/340436258_Movie_recommendation_system_with_Collaborative_Filtering_using_K-NN
- https://le-james94.medium.com/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223
- https://iopscience.iop.org/article/10.1088/1742-6596/1964/4/042081
- https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea

2. Overview of the final process

2.1 Dataset and Domain

2.1. 1 Dataset

Dataset Source: https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

This dataset consists of the following files:

- **movies_metadata.csv:** The main Movies Metadata file. Contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.

- **keywords.csv:** Contains the movie plot keywords for our MovieLens movies. Available in the form of a stringified JSON Object.

- **credits.csv:** Consists of Cast and Crew Information for all our movies. Available in the form of a stringified JSON Object.

- **ratings.csv:** Consists of ratings,timestamp for all our movies.

2.1.2 Data Dictionary

Here is the overview of all our variables along with the data-type,description and whether it is required for our model or not.

Column	Data type	Description	Required?
adult	object	Adult content or not	Y
budget	float64	Budget of the film	N
homepage	object	Homepage of film	N
id	object	Movie id	Y
imdb_id	int64	IMBD id	Y
original_language	object	Original Language	Y
original_title	object	Original Title	Y
overview	object	Summary/Overview	Y
popularity	float64	Popularity of movie	Y
production_countries	datetime[ns]	Production country	Y
release_date	float64	Release date of movie	N
revenue	float64	Revenue generated	N
runtime	float64	Overall Runtime	N

status	object	Release or under production	Y
title	object	Title of the movie	N
vote_average	float64	Voting Average	Y
vote_count	float64	No. of users voted	Y
genre_name	object	Type of Genre	Y
collection_name	object	Movie's collection name	Y
production_companies_name	object	Production company name	Y
rating	float64	Average rating	Y
cast	object	Cast of movie	Y
crew	object	Crew of movie	Y

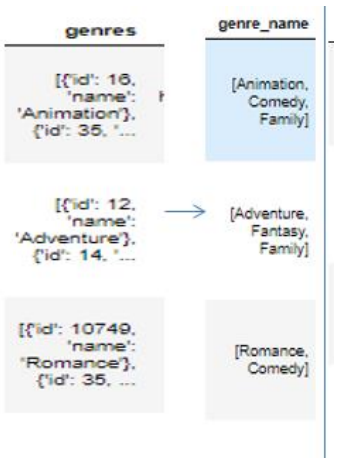
2.1.3 Variable categorization (count of numeric and categorical)

Here we are dealing with different types of variable that can be categorized as Numerical and Categorical.

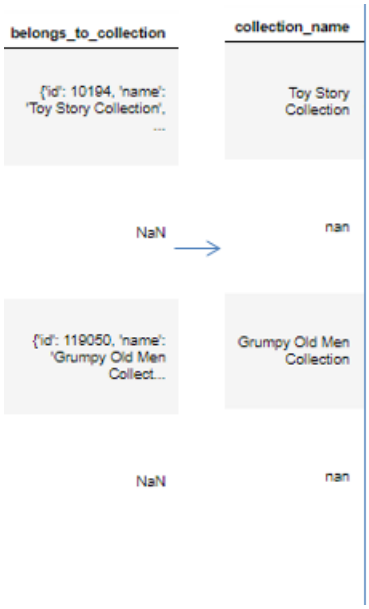
Numerical	Categorical
budget	adult
id	homepage
imdb_id	original_language
popularity	original_title
release_date	overview
revenue	production_countries
runtime	status
vote_average	title
vote_count	video
rating	genre_name
	collection_name
	production_companies_name
	cast
	crew

2.2 Data Cleaning

1. **genre_name:** We have extracted the name from genres column given below which given as a form of List of Dictionary and the List of Dictionary was itself in the form of string. We
   Used "literal_eval" function given in python in "ast" module and store in the column.



2. **collection_name:** We have extracted the name belongs_to_collection column given below which given as a form of List of Dictionary and the List of Dictionary was itself in the form of string. We used “literal_eval” function given in python in “ast” module and store in the column.

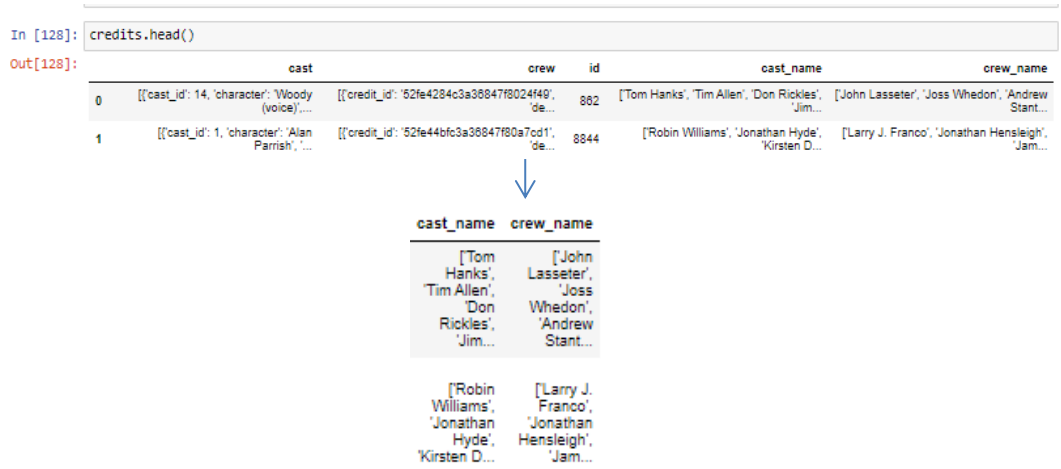


3. **keywords:** We have extracted the name keywords column given below which given as a form of List of Dictionary and the List of Dictionary was itself in the form of string. We used “literal_eval” function given in python in “ast” module to extract and store as the list.

	id	keywords
0	862	[{'id': 931, 'name': 'jealousy'}, {'id': 4290,...
1	8844	[{'id': 10090, 'name': 'board game'}, {'id': 1...
2	15602	[{'id': 1495, 'name': 'fishing'}, {'id': 12392...
3	31357	[{'id': 818, 'name': 'based on novel'}, {'id':...
4	11862	[{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n...
...	...	...
31619	84419	[{'id': 9748, 'name': 'revenge'}, {'id': 9826,...
31620	390959	[{'id': 224180, 'name': 'blair witch'}]
31621	289923	[{'id': 616, 'name': 'witch'}, {'id': 2035, 'n...
31622	439050	[{'id': 10703, 'name': 'tragic love'}]
31623	111109	[{'id': 2679, 'name': 'artist'}, {'id': 14531,...

	id	keywords
0	862	[jealousy, toy, boy, friendship, friends, riva...
1	8844	[board game, disappearance, based on children'...
2	15602	[fishing, best friend, duringcreditsstinger, o...

4. **Credits and Crew:** We have extracted the name credits column given below which given as a form of List of Dictionary and the List of Dictionary was itself in the form of string. We used “literal_eval” function given in python in “ast” module to extract and store as the list.



2.2.1 Final Cleaned Dataset

After cleaning and removing the unwanted columns in the dataset, we finally have the dataset with the following column names:

```
Index(['adult', 'id', 'original_language', 'original_title', 'overview',
       'popularity', 'release_date', 'revenue', 'runtime', 'status',
       'genre_name', 'collection_name', 'production_companies_name',
       'keywords', 'cast_name', 'crew_name', 'vote_average'],
      dtype='object')
```

2.2.2 Cleaned Data Dictionary – Variable Categorisation:

Numerical	Categorical
budget	adult
id	original_language
popularity	original_title
release_date	overview
vote_average	collection_name
	production_companies_name
	keywords
	cast_name
	crew_name

2.3 Data Exploration

2.3.1 Data Types

We can see that there are around 45000 rows and there are 16 columns in our final dataset. We have 4 numerical columns and some categorical columns as well.

```
In [225]: 1 dataframe_copy.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 45463 entries, 0 to 45462
Data columns (total 17 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   adult                      45463 non-null  object
 1   id                         45463 non-null  int64
 2   original_language          45452 non-null  object
 3   original_title             45463 non-null  object
 4   overview                   44509 non-null  object
 5   popularity                 45460 non-null  float64
 6   release_date               45376 non-null  datetime64[ns]
 7   revenue                    45460 non-null  float64
 8   runtime                    45203 non-null  float64
 9   status                     45379 non-null  object
 10  genre_name                 45463 non-null  object
 11  collection_name            45463 non-null  object
 12  production_companies_name  45463 non-null  object
 13  keywords                   31110 non-null  object
 14  cast_name                  45462 non-null  object
 15  crew_name                  45462 non-null  object
 16  vote_average               45460 non-null  float64
dtypes: datetime64[ns](1), float64(4), int64(1), object(11)
memory usage: 6.2+ MB
```

2.3.2 Null Value Detection

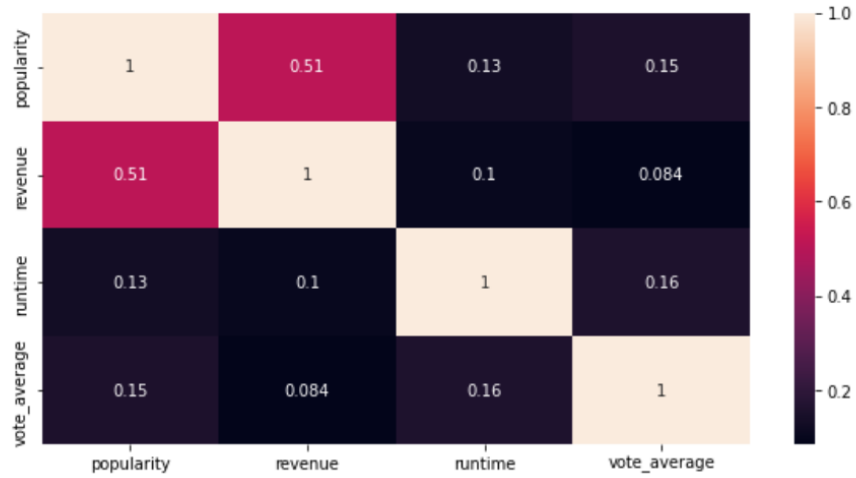
Our dataset is having some null values especially in the Keywords columns. We didn't remove it since we thought that this can affect the final model.

```
In [263]: 1 dataframe_copy.isnull().sum()

Out[263]: adult                         0
          id                            0
          original_language            11
          original_title                0
          overview                    954
          popularity                    3
          release_date                 87
          revenue                       3
          runtime                     260
          status                       84
          genre_name                    0
          collection_name               0
          production_companies_name     0
          keywords                  14353
          cast_name                     1
          crew_name                     1
          vote_average                  3
          dtype: int64
```

2.3.3 Correlation between numeric variables

Below is the Heatmap for all the numerical columns in the data set:

```
In [305]:
1 sns.heatmap(df_ss_num.iloc[:,1:].corr(),annot=True)
2 plt.show()
```



We can see that there is no very strong relationship between any column. Although we can see there is moderate correlation between the revenue and popularity since there is chance that some of movie because of high popularity can generate high revenue or vice versa but not always.

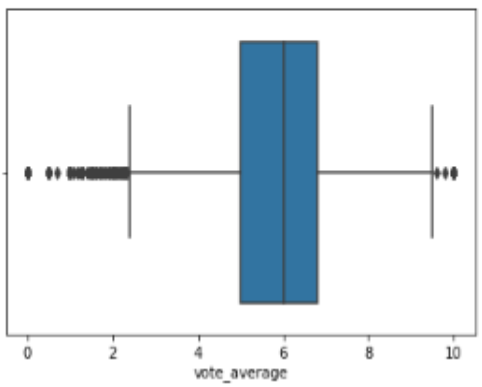
2.3.4 Outlier Analysis & Treatment

Since in our recommendation system, we'll consider 'vote average' as an important feature in numeric dataset, we'll remove the outliers for that.

```
sns.boxplot(dataframe['vote_average'])
```

```
C:\Users\Priyanka\anaconda3\lib\site-packages\seab
d arg: x. From version 0.12, the only valid positi
keyword will result in an error or misinterpretati
  warnings.warn(
```

```
<AxesSubplot:xlabel='vote_average'>
```




```
#vote_average outliers
Q1=dataframe['vote_average'].quantile(0.25)
Q3=dataframe['vote_average'].quantile(0.75)
IQR=Q3-Q1
print(Q1)
print(Q3)
print(IQR)
Lower_Whisker = Q1-1.5*IQR
Upper_Whisker = Q3+1.5*IQR
print(Lower_Whisker, Upper_Whisker)
```

```
5.0
6.8
1.7999999999999998
2.3000000000000003 9.5
```

```
out_dataframe_vote_avg = dataframe[dataframe['vote_average']< Upper_Whisker]
out_dataframe_vote_avg = dataframe[dataframe['vote_average']>Lower_Whisker]
```

```
out_dataframe = out_dataframe[out_dataframe['vote_average']< Upper_Whisker]
out_dataframe = out_dataframe[out_dataframe['vote_average']>Lower_Whisker]
```

```
len(out_dataframe_vote_avg)
```

```
42049
```

```
len(out_dataframe)
```

```
34626
```

2.4 Algorithms overview

- The Content-based Filtering is done using Count Vectorizer and Cosine Similarity and using TfidfVectorizer and Cosine Similarity.

- The Collaborative Filtering is done based on either User-based or Item-based parameters.

- In Hybrid Filtering, we merge the two recommended techniques Content-based and Collaborative filtering to get the better result.

- The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) - calculating the distance between points on a graph.

3. Step-by-step walk through of the solution

3.1 Data Cleaning walkthrough

We have already seen the results of data cleaning at *2.2 Data Cleaning*. We'll see the code on the data cleaning.

1. Genre:

```
]: final_names = []
for i in range(len(metadata)):
    ini_list=metadata.genres[i]
    res = ast.literal_eval(ini_list)
    if isinstance(res, list):
        final_names.append([i['name'] for i in res])
    elif isinstance(res,dict):
        final_names.append([i["name"]])

final_names = {"name":final_names}
genre_list=pd.DataFrame(final_names)
print(genre_list)
```

```
                                name
0       [Animation, Comedy, Family]
1      [Adventure, Fantasy, Family]
2                  [Romance, Comedy]
3          [Comedy, Drama, Romance]
4                          [Comedy]
...                             ...
45461              [Drama, Family]
45462                      [Drama]
45463     [Action, Drama, Thriller]
45464                             []
45465                             []

[45466 rows x 1 columns]
```

2. Cast:

```
for i in range(0,len(credits)):
    ini_list =credits.cast[i]
    res = ast.literal_eval(ini_list)
    #print(res)
    #str2=str1
    str1=''
    li=[]
    #print(len(res))
    for j in range(0,len(res)):
        res1=ast.literal_eval(str(res[j]))
        str1=str1+','+res1['name']
        #df = pd.DataFrame([str1], columns=['string_values'])
        li.append(res1['name'])
    #print(i)
    #str2=str1
    df1=df1.append([str(li)])
```

```
df1.rename(columns={0:'cast_name'},inplace=True)
```

```
credits=credits.reset_index(drop=True).merge(df1.reset_index(drop=True), left_index=True, right_index=True)
```

```
credits.drop(columns=['Cast'],inplace=True)
```

```
credits.head()
```

	cast	crew	id	cast_name
0	[{'cast_id': 14, 'character': 'Woody (voice)',...	[{'credit_id': '52fe4284c3a36847f8024f49', 'de...	862	['Tom Hanks', 'Tim Allen', 'Don Rickles', 'Jim...
1	[{'cast_id': 1, 'character': 'Alan Parrish', '...	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'de...	8844	['Robin Williams', 'Jonathan Hyde', 'Kirsten D...
2	[{'cast_id': 2, 'character': 'Max Goldman', 'c...	[{'credit_id': '52fe466a9251416c75077a89', 'de...	15602	['Walter Matthau', 'Jack Lemmon', 'Ann-Margret...
3	[{'cast_id': 1, 'character': "Savannah 'Vannah...	[{'credit_id': '52fe44779251416c91011acb', 'de...	31357	['Whitney Houston', 'Angela Bassett', 'Loretta...
4	[{'cast_id': 1, 'character': 'George Banks', '...	[{'credit_id': '52fe44959251416c75039ed7', 'de...	11862	['Steve Martin', 'Diane Keaton', 'Martin Short...

3. Crew:

```
for i in range(0,len(credits)):
    ini_list =credits.crew[i]
    res = ast.literal_eval(ini_list)
    #print(res)
    #str2=str1
    str1=''
    li=[]
    #print(i)
    for j in range(0,len(res)):
        res1=ast.literal_eval(str(res[j]))
        str1=str1+','+res1['name']
        #df = pd.DataFrame([str1], columns=['string_values'])
        li.append(res1['name'])
    #print(str1)
    #str2=str1
    df2=df2.append([str(li)])
```

```
df2.drop(columns=['crew'],inplace=True)
```

```
df2.head()
```

	crew_name
0	['John Lasseter', 'Joss Whedon', 'Andrew Stant...
0	['Larry J. Franco', 'Jonathan Hensleigh', 'Jam...
0	['Howard Deutch', 'Mark Steven Johnson', 'Mark...
0	['Forest Whitaker', 'Ronald Bass', 'Ronald Bas...
0	['Alan Silvestri', 'Elliot Davis', 'Nancy Meye...

4. Keywords:

```
list1 =  df_keywords['keywords'].tolist()
for i in range(len(list1)) :
    list1[i] = ast.literal_eval(list1[i])
```

```
df_keywords['keywords'] = list1
```

```
C:\Users\Priyanka\AppData\Local\Temp/ipykernel_12416/813982069.py:1
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/par
rsus-a-copy
  df_keywords['keywords'] = list1
```

```
df_keywords_copy = df_keywords.copy()
for i in range(0,len(df_keywords['keywords'])):
    list1 = []
    for j in range(len(df_keywords['keywords'].iloc[i])):
        list1.append(df_keywords.iloc[i]['keywords'][j]['name'])
    df_keywords_copy['keywords'].iloc[i] = list1
```

```
C:\Users\Priyanka\anaconda3\lib\site-packages\pandas\core\indexing.
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/par
rsus-a-copy
  self._setitem_single_block(indexer, value, name)
```

```
df_keywords_copy.head(3)
```

	id	keywords
0	862	[jealousy, toy, boy, friendship, friends, riva...
1	8844	[board game, disappearance, based on children'...
2	15602	[fishing, best friend, duringcreditsstinger, o...

3.2 Cleaned Overall Dataset

After cleaning the columns in the dataset, we have selected specific columns that will be useful for us to predict/recommend movies based on our model.

```
In [166]: dataframe = pd.DataFrame().assign(adult=main1['adult'],
id=main1['id'],original_language=main1['original_language'],original_title=main1['original_title'],
overview=main1['overview'],popularity=main1['popularity'],
release_date=main1['release_date'],revenue=main1['revenue'],runtime=main1['runtime'],
status=main1['status'],
genre_name=main1['genre_name'],collection_name=main1['collection_name'],
production_companies_name=main1['production_companies_name'],keywords=main1['keywords'],cast_name=main1['cast_name'],
crew_name=main1['crew_name'],vote_average=main1['vote_average'])
```

```
: dataframe.isnull().sum()

: adult                          0
  id                             0
  original_language             11
  original_title                 0
  overview                     954
  popularity                     3
  release_date                  87
  revenue                        3
  runtime                      260
  status                        84
  genre_name                     0
  collection_name                0
  production_companies_name      0
  keywords                   14353
  cast_name                      1
  crew_name                      1
  vote_average                   3
  dtype: int64
```

3.3 Feature Selection and Scaling

3.3.1 Feature Encoding

Encoding all the categorical columns and then merge them with numerical columns to get a overall dataframe that can be used for model.

```
new_encoded_values = pd.get_dummies(data=dataframe_copy, prefix=None, prefix_sep='_', dummy_na=False, columns=['adult','status'])
new_encoded_values.head(2)
```

	id	original_language	original_title	overview	popularity	release_date	revenue	runtime	genre_name	collection_name	...	crew_name	vote_average
0	862	en	Toy Story	Led by Woody, Andy's toys live happily in his ...	21.946943	1995-10-30	373554033.0	81.0	[Animation, Comedy, Family]	Toy Story Collection	...	['John Lasseter', 'Joss Whedon', 'Andrew Stant...	7.7
1	8844	en	Jumanji	When siblings Judy and Peter discover an encha...	17.015539	1995-12-15	262797249.0	104.0	[Adventure, Fantasy, Family]	nan	...	['Larry J. Franco', 'Jonathan Hensleigh', 'Jam...	6.9

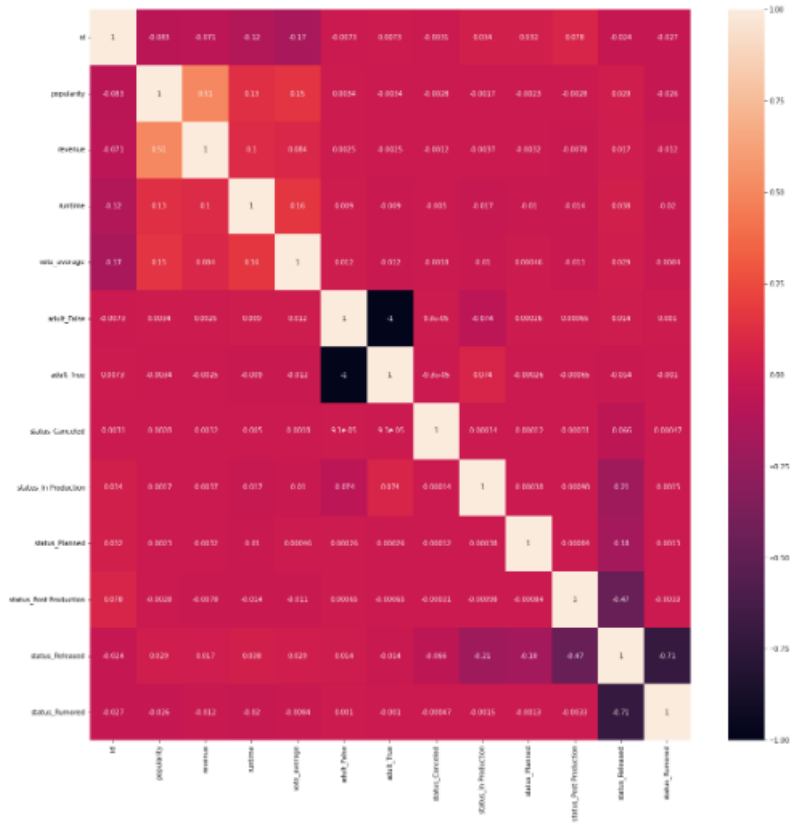
2 rows × 23 columns

3.3.2 Muticollinearity using Heatmap and VIF

Finding the columns that have high correlation with other columns using Heatmap and Variation Inflation Factor

3.3.2.1 Heatmap

```
In [211]: correlated = new_encoded_values.corr()
          fig, ax = plt.subplots(figsize=(20,20))
          sns.heatmap(correlated,annot=True)
          plt.show()
          # write observations for categorical variables correlation based on eigen values
```



```
In [ ]: # Popularity , revenue , vote_average -> VIF to check multi-collinearity
```

3.3.2.2: Variation Inflation Factor

8.2.Variance_Inflation_Factor ¶

In [218]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    return(vif)


X = data_numeric_val.iloc[:,:-1]
calc_vif(X)
```

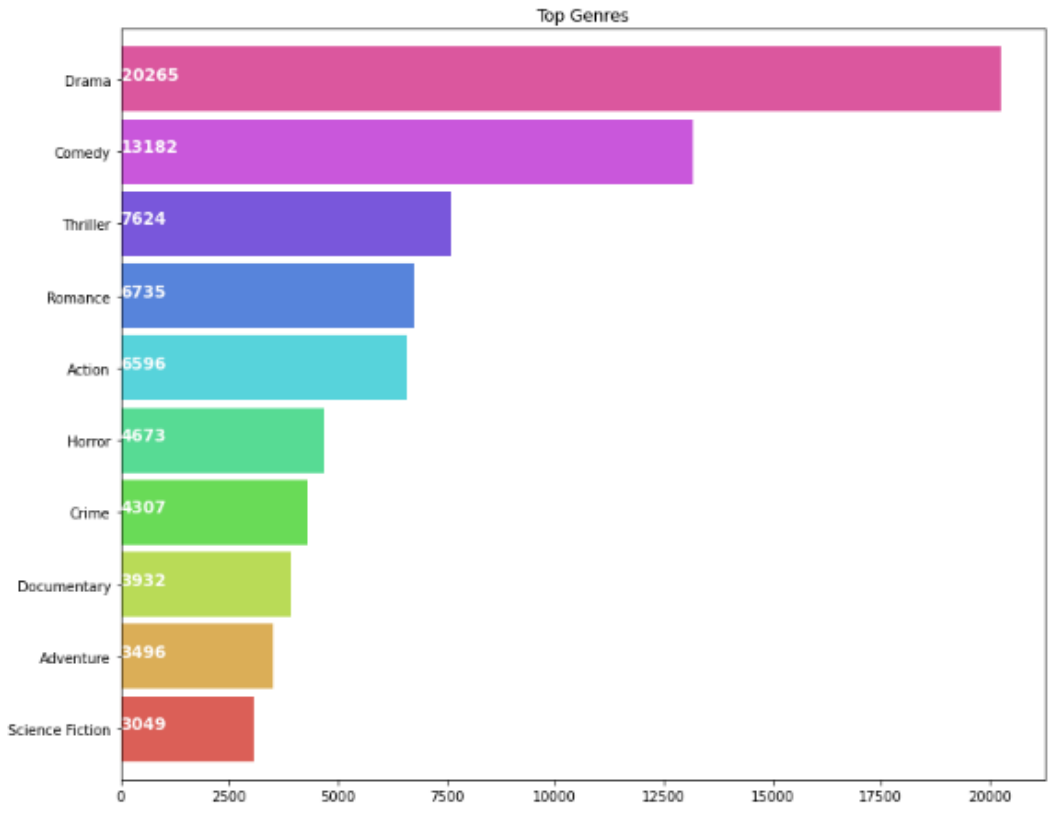
Out[218]:

	variables	VIF
0	id	1.594995
1	popularity	1.663890
2	revenue	1.385580
3	runtime	1.887466

From the VIF and Heatmap, we can see that the variables are not that much correlated.

3.4 Data Visualizations

3.4.1 Visualization on Genre



Inference:

- Drama genre has the highest number of movies in our dataset, followed by Comedy and Thriller.
- Sci-Fi has the least number of movies in our dataset

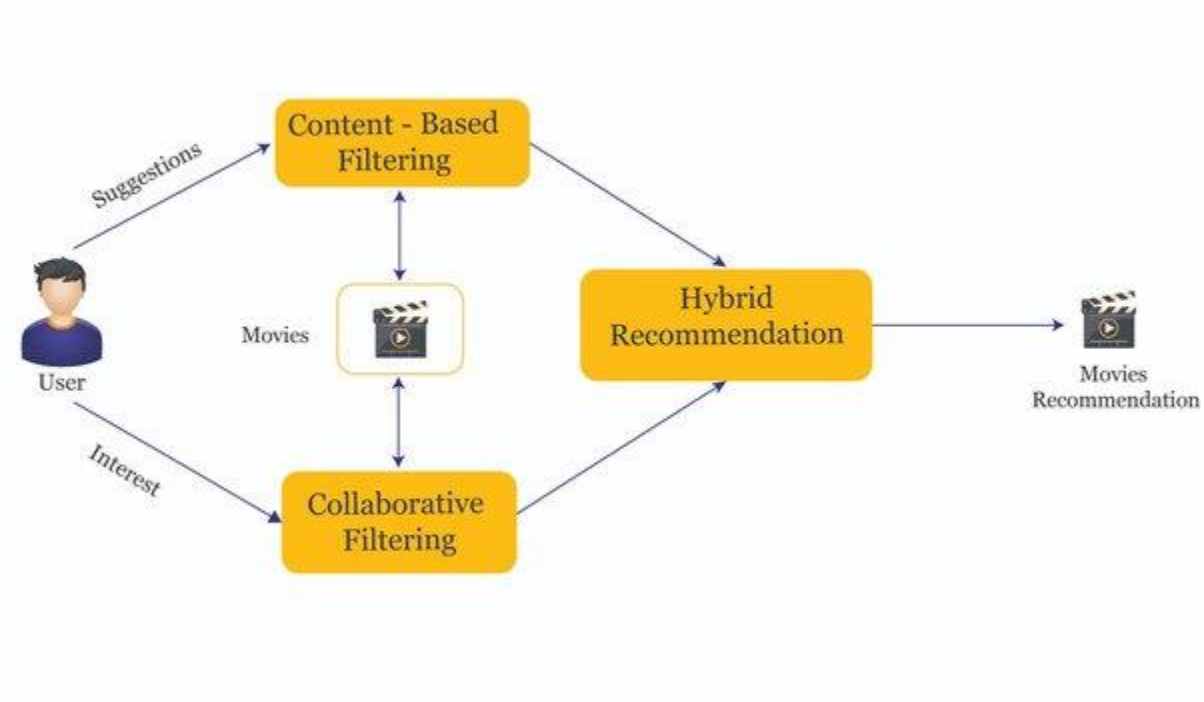
3.5 Performing Recommendation Techniques

The Recommendation Techniques used in this project are:

- K-Nearest Neighbours Supervised Algorithm
- Hybrid Filtering

5. Model Building and evaluation

5.1. Hybrid Filtering



4.1.1. Content – Based Recommendation

Content-Based Recommendations systems are the systems that look for similarity before recommending something.This is all done through content-based systems. The similarity of different movies is computed to the one you are currently watching and all the similar movies are recommended to us.

Content Based Recommendation System: It uses attributes such as genre, description, actors, etc. for movies, to make suggestions for the users. The intuition behind this sort of recommendation system is that if a user liked a particular movie or show, he/she might like a movie or a show similar to it.

- **Cosine Similarity:-** This type of metric is used to compute the similarity textual data. We convert these textual data in the form of vectors and check for cosine angle between those two vectors if the angle between them is 0. It means they are similar or else they are not. Most used similarity measures when we talk about the similarity between any textual content.

```
In [38]: get_recommendations('The Godfather').head(10)

Out[38]: 982              The Godfather: Part II
         1590            The Godfather: Part III
         969                      Apocalypse Now
         2986                   The Conversation
         3685                    The Cotton Club
         775     Around the World in Eighty Days
         4486                  One from the Heart
         1679                       The Outsiders
         5827                         Rumble Fish
         2364                       South Pacific
         Name: original_title, dtype: object
```

```
In [39]: get_recommendations('The Dark Knight').head(10)

Out[39]: 7971                                The Dark Knight Rises
         6174                                       Batman Begins
         6575                                        The Prestige
         8750                                      Jurassic World
         3323                                          15 Minutes
         6188                                   War of the Worlds
         4505                                           Daredevil
         5564                                          Collateral
         3466                        A.I. Artificial Intelligence
         4903    Master and Commander: The Far Side of the World
         Name: original_title, dtype: object
```

4.1.2. Collaborative Filtering

To address some of the limitations of content-based filtering, collaborative filtering uses *similarities between users and items simultaneously* to provide recommendations. This allows for serendipitous recommendations; that is, collaborative filtering models can recommend an item to user A based on the interests of a similar user B. Furthermore, the embeddings can be learned automatically, without relying on hand-engineering of features.

Collaborative filtering filters information by using the interactions and data collected by the system from other users. It's based on the idea that people who agreed in their evaluation of certain items are likely to agree again in the future.

The concept is simple: when we want to find a new movie ,we'll often ask our friends for recommendations. Naturally, we have greater trust in the recommendations from friends who share tastes similar to our own.

Most collaborative filtering systems apply the so-called **similarity index-based technique**. In the neighborhood-based approach, a number of users are selected based on their similarity to the active user. Inference for the active user is made by calculating a **weighted average of the ratings** of the selected users.

Collaborative-filtering systems focus on the relationship between users and items. The similarity of items is determined by the similarity of the ratings of those items by the users who have rated both items.

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. Collaborative Filtering is based on the idea that users similar to a me can be used to predict how much I will like a particular product or service those users have used/experienced but I have not.

Using the K-Fold, where K=5:

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8940  0.8965  0.8988  0.9030  0.8882  0.8961  0.0049
MAE (testset)     0.6892  0.6910  0.6919  0.6935  0.6845  0.6900  0.0031
Fit time          0.97    0.97    0.97    0.98    0.99    0.98    0.01
Test time         0.11    0.12    0.22    0.12    0.11    0.14    0.04

5]: {'test_rmse': array([0.89402443, 0.89651112, 0.89880489, 0.90295496, 0.8881786 ]),
     'test_mae': array([0.68919363, 0.69100958, 0.69188443, 0.69349738, 0.68451132]),
     'fit_time': (0.965803861618042,
      0.9685196876525879,
      0.9685194492340088,
      0.9841151237487793,
      0.9882588386535645),
     'test_time': (0.1093449592590332,
      0.12494158744812012,
      0.21869611740112305,
      0.12496781349182129,
      0.10934567451477051)}
```

4.1.3. Hybrid Filtering

In this section, we built a simple hybrid recommender that brings together techniques we have implemented in the content based and collaborative filter based engines. This is how it will work:

- Input: User ID and the Title of a Movie
- Output: Similar movies sorted on the basis of expected ratings by that particular user.

The movie recommendation system is a hybrid filtering system that performs both collaborative and content-based filtering of data to provide recommendations to users regarding movies. The system conforms to a different approach where it seeks the similarity of users among others clustered around the various genres and utilizes his preference of movies based on their content in terms of genres as the deciding factor of the recommendation of the movies to them.

The system is based on the belief that a user rates movies in a similar fashion to other users that harbor the same state as the current user and is also affected by the other activities (in terms of rating) he performs with other movies.

```
hybrid(500, 'Father of the Bride Part II')
```

51]:

	original_title	vote_average	genre_name	id	est
8978	The Intern	7.1	[Comedy]	257211	3.455421
6595	The Holiday	6.7	[Comedy, Romance]	1581	3.351483
1675	Dead Men Don't Wear Plaid	6.5	[Comedy, Mystery]	9442	3.281626
3407	The World According to Garp	6.9	[Comedy]	11307	3.239328
4308	The Four Seasons	6.3	[Comedy, Drama, Romance]	25113	3.220658
5038	Something's Gotta Give	6.3	[Drama, Comedy, Romance]	6964	3.217175
2180	Drop Dead Gorgeous	6.5	[Comedy]	10490	3.216455
3223	What Women Want	6.1	[Comedy, Romance]	3981	3.198183
8466	Philomena	7.4	[Drama]	205220	3.151926
4833	Bugsy Malone	6.5	[Drama, Action, Comedy, Music, Family]	8446	3.151128

```
Enter the user6789
Enter the movieIt Takes Two
Enter the user45
Enter the movieLe confessionnal
Enter the user4987
Enter the movieLa Haine
Enter the user1
Enter the movieCasper
Enter the user678
Enter the movieThe War
```

[58]: 0.6085634296770952

Limitations:

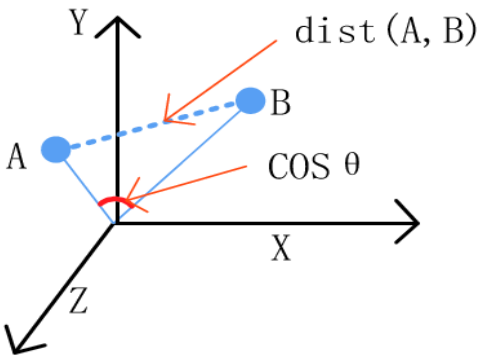
- Our content based engine suffers from some severe limitations. It is only capable of suggesting movies which are *close* to a certain movie. That is, it is not capable of capturing tastes and providing recommendations across genres.

- 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.

4.2 K-Nearest Neighbours Supervised Algorithm

KNN (**K-Nearest Neighbors**) is a simple machine learning algorithm that is used for movie recommendation systems. The basic idea behind this algorithm is that the movie that is most similar to a user's past movie preferences will be the one they are most likely to enjoy.

To implement a KNN movie recommendation system, you would start by gathering a dataset of movies and their features (e.g., genre, keywords, cast_name, etc.). Then, you would use a similarity metric (e.g., cosine similarity) to compare the features of the movies in your dataset and determine the "closest" movies to a user's past preferences. Finally, you would recommend the top K movies (where K is a user-defined parameter) that are most similar to the user's past preferences.



Limitations:

1. It cannot handle the "cold start" problem (when a user has not yet rated any movies) well.

Output:

The output displays the selected movie you like and 10 similar movies, it’s predicted ratings and **RMSE** metric.

Movie Recommendation based on ‘Toy Story’:

```
Selected Movie:  Toy Story

The Recommended Movies:

Toy Story 3 | Genres: 'Animation','Family','Comedy'
Toy Story 2 | Genres: 'Animation','Comedy','Family'
The Tangerine Bear: Home in Time for Christmas! | Genres: 'Animation','Family'
Meet the Robinsons | Genres: 'Animation','Comedy','Family'
Toy Story That Time Forgot | Genres: 'Animation','Family'
Creature Comforts | Genres: 'Animation','Comedy','Family'
Bébé's Kids | Genres: 'Animation','Comedy','Family'
The SpongeBob SquarePants Movie | Genres: 'Animation','Comedy','Family'
Toy Story of Terror! | Genres: 'Animation','Comedy','Family'
The Madagascar Penguins in a Christmas Caper | Genres: 'Animation','Comedy','Family'
Superstar Goofy | Genres: 'Animation','Comedy','Family'
Lorenzo | Genres: 'Animation','Comedy','Family'


The predicted rating for Toy Story is: 6.691667
The actual rating for Toy Story is 7.700000
**************************************************************************************************
```

Movie Recommendation based on ‘Jumanji’:

```
Selected Movie:  Jumanji

The Recommended Movies:

Escape to Witch Mountain | Genres: 'Adventure','Family','Fantasy'
Hook | Genres: 'Adventure','Fantasy','Comedy','Family'
Gekijô ban Tottoko Hamutarô: Hamu hamu hamu~jya! Maboroshi no prinsesu | Genres: 'Fantasy','Adventure','Animation','Family
Charlie, the Lonesome Cougar | Genres: 'Adventure','Family'
Kismet | Genres: 'Adventure','Fantasy'
A Trip to Mars | Genres: 'Fantasy','Adventure'
Captain Nemo and the Underwater City | Genres: 'Adventure','Family'
Further Adventures of the Wilderness Family | Genres: 'Adventure','Family'
Sovsem Propashchiy | Genres: 'Family','Adventure'
VeggieTales: LarryBoy & The Bad Apple | Genres: 'Adventure','Family'
VeggieTales: The Wonderful Wizard of Ha's | Genres: 'Adventure','Family'
La Cité des Enfants Perdus | Genres: 'Fantasy','ScienceFiction','Adventure'


The predicted rating for Jumanji is: 4.775000
The actual rating for Jumanji is 6.900000
**************************************************************************************************
```

Movie Recommendation based on ‘Grumpier Old Men’:

```
Selected Movie:  Grumpier Old Men

The Recommended Movies:

Town & Country | Genres: 'Comedy','Romance'
Cactus Flower | Genres: 'Comedy','Romance'
A New Leaf | Genres: 'Comedy','Romance'
A Guide for the Married Man | Genres: 'Comedy','Romance'
White on Rice | Genres: 'Comedy','Romance'
What's Up, Scarlet? | Genres: 'Comedy','Romance'
Made For Each Other | Genres: 'Comedy','Romance'
Beauty and the Boss | Genres: 'Comedy','Romance'
House Calls | Genres: 'Comedy','Romance'
Act Like You Love Me | Genres: 'Comedy','Romance'
Shampoo | Genres: 'Comedy','Drama','Romance'
42nd Street | Genres: 'Music','Comedy','Romance'


The predicted rating for Grumpier Old Men is: 6.050000
The actual rating for Grumpier Old Men is 6.500000
**************************************************************************************************
```

Movie Recommendation based on ‘Waiting to Exhale’:

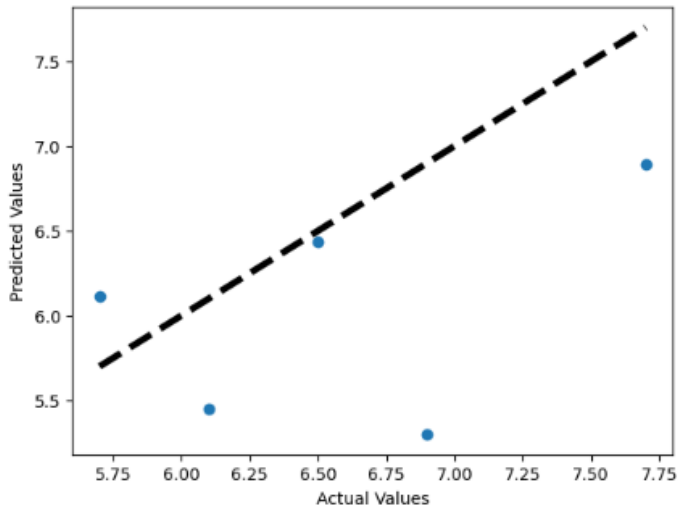
```
Selected Movie:  Waiting to Exhale

The Recommended Movies:

Dreamworld | Genres: 'Comedy','Romance','Drama'
Hello Lonesome | Genres: 'Drama','Comedy','Romance'
Corrina, Corrina | Genres: 'Comedy','Drama','Family','Romance'
Voy a explotar | Genres: 'Drama','Comedy','Romance','Foreign'
Boy | Genres: 'Drama','Comedy','Romance','Foreign'
Boys on the Side | Genres: 'Comedy','Drama'
The Telephone | Genres: 'Drama','Comedy'
Cowards Bend the Knee | Genres: 'Drama','Romance'
La estrategia del caracol | Genres: 'Comedy','Drama'
Lieksa! | Genres: 'Comedy','Drama'
The Trouble with Dee Dee | Genres: 'Comedy','Drama'
The Village Barbershop | Genres: 'Comedy','Drama'


The predicted rating for Waiting to Exhale is: 5.958333
The actual rating for Waiting to Exhale is 6.100000
**************************************************************************************************
```

Movie Recommendation based on 'Father of the Bride Part II':

```
Selected Movie:  Father of the Bride Part II

The Recommended Movies:

審死官 | Genres: 'Comedy'
Sour Grapes | Genres: 'Comedy'
The Jerk | Genres: 'Comedy'
The Out-of-Towners | Genres: 'Comedy'
Bowfinger | Genres: 'Comedy'
Babes in Toyland | Genres: 'Comedy'
The Original Kings of Comedy | Genres: 'Comedy'
...
The predicted rating for Father of the Bride Part II is: 6.041667
The actual rating for Father of the Bride Part II is 5.700000
****************************************************************************************************
RMSE value :  1.0836666154003887
```

- **Grapical representation using Actual vs Predicted Values:**

```
In [36]: # Grapical representation using Actual vs Predicted Values


         def plot_actual_vs_predicted(actual, predicted):
             plt.scatter(actual, predicted)
             plt.xlabel("Actual Values")
             plt.ylabel("Predicted Values")
             plt.plot([actual.min(), actual.max()], [actual.min(), actual.max()], 'k--', lw=4)
             plt.show()

         plot_actual_vs_predicted(actual_rating, predicted_avg_rating_arr)
```



- **Hyperparameter Tuning on the recommendation task :**

In KNN, one of the most important hyperparameters is the value of k, which determines the number of nearest neighbors used to make predictions.

The use of applying different k values with RMSE (Root Mean Squared Error) for movie recommendation using KNN (K-Nearest Neighbors) is to evaluate and compare the performance of the recommendation system with different values of k. RMSE is a common evaluation metric used in recommendation systems to measure the difference between the predicted ratings and the actual ratings. By calculating the RMSE for different values of k,

the recommendation system can determine the value of k that gives the lowest RMSE and thus, the best performance.

The optimal value of k is the one that gives the **lowest RMSE**, indicating that the predicted ratings are closest to the actual ratings. By using RMSE, the recommendation system can make an objective and quantitative assessment of its performance and find the best value of k for the movie recommendation task.

Thus, Eventhough K value is randomly picked in HPT, There is no change in the RMSE value,it appears to be same for different K values. It means that, the model performance does not change significantly as you increase or decrease the K value.

6. Feature Engineering and Selection of models

1. **For KNN model** ,we need to convert our data set into numerical data. We need to genre, cast_name, keywords columns for our KNN model so we need to convert them into binary form. These columns are in the form of list of string.

```
[9]: id                                                    862
     original_title                                  Toy Story
     genre_name                  ['Animation', 'Comedy', 'Family']
     cast_name        ['Tom Hanks', 'Tim Allen', 'Don Rickles', 'Jim...
     keywords         ['jealousy', 'toy', 'boy', 'friendship', 'frie...
     vote_average                                           7.7
     Name: 0, dtype: object
```

- For genre columns we try to find the distinct genre present in the genre columns, and we store them in form of list which is given below.

```
['Animation',
 'Comedy',
 'Family',
 'Adventure',
 'Fantasy',
 'Romance',
 'Drama',
 'Action',
 'Crime',
 'Thriller']
```

- We use above list to convert the value from string to binary given below. Here 1 mean the movie having that genre whereas 0 mean the movie does not have that genre.

```
1 df_knn_data['genres_bin'].head(3)
```

```
0    [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1    [0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2    [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
Name: genres_bin, dtype: object
```

- Similarly, we perform the same for keywords and cast_name and result given below:

```
1  df_knn_data['words_bin'].head(3)
```

```
0    [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, ...
1    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, ...
2    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
Name: words_bin, dtype: object
```

```
|:  1  df_knn_data['cast_bin'].head()
```

```
|: 0    [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
   1    [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
   2    [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
   3    [0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
   4    [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, ...
   Name: cast_bin, dtype: object
```

- Below is the final column after doing feature engineering.

```
9]:  df_data.columns
```

```
'9]: Index(['id', 'original_title', 'genre_name', 'cast_name', 'keywords',
        'vote_average', 'genres_bin', 'words_bin', 'cast_bin', 'new_id'],
       dtype='object')
```

2. Hybrid Filtering:

- We need to clean the word and remove duplicate words for column keywords.

t[2]:

	adult	id	original_language	original_title	overview	popularity	release_date	revenue	runtime	status	genre_name	collection_name	production_companies_name	
0	False	862	en	Toy Story	Led by Woody, Andy's toys live happily in his ...	21.946943	1995-10-30	373554033.0	81.0	Released	['Animation', 'Comedy', 'Family']	Toy Story Collection	['Pixar Animation Studios']	['j
1	False	8844	en	Jumanji	When siblings Judy and Peter discover an encha...	17.015539	1995-12-15	262797249.0	104.0	Released	['Adventure', 'Fantasy', 'Family']	NaN	['TriStar Pictures', 'Teitler Film', 'Intersco...	['di "b

keywords	cast_name	crew_name	vote_average
['jealousy', 'toy', 'boy', 'friendship', 'frie...	['Tom Hanks', 'Tim Allen', 'Don Rickles', 'Jim...	['John Lasseter', 'Joss Whedon', 'Andrew Stant...	7.7
['board game', 'disappearance', "based on chil...	['Robin Williams', 'Jonathan Hyde', 'Kirsten D...	['Larry J. Franco', 'Jonathan Hensleigh', 'Jam...	6.9

- After cleaning and removing the duplicate words and store in new column.

keywords_x	cast_name	crew_name	vote_average	keywords_y	keywords)y
['jealousy', 'toy', 'boy', 'friendship', 'frie...	['Tom Hanks', 'Tim Allen', 'Don Rickles', 'Jim...	['John Lasseter', 'Joss Whedon', 'Andrew Stant...	7.7	[jealousi, toy, boy, friendship, friend, rival...	[jealousi, toy, boy, friendship, friend, rival...
['board game', 'disappearance', "based on chil...	['Robin Williams', 'Jonathan Hyde', 'Kirsten D...	['Larry J. Franco', 'Jonathan Hensleigh', 'Jam...	6.9	[board gam, disappear, based on children's boo...	[boardgam, disappear, basedonchildren'sbook, n...

3. Comparison and Selection of Model

After comparing our model, we can see that:

1. RMSE value for KNN is around 1.083 (based to average voting)
2. RMSE value using collaborative filtering is around 0.9 (based on rating)
3. RMSE for Hybrid filtering is around 0.6 (based on rating)

But for KNN we are not considering RMSE value (since it is not giving any clear view) instead of that we are trying to see what movie it is trying to recommend based on genre. We can see that it is very close to the expected result.

4. Results and discussion

From above comparison we can see that all the models are predicting almost same movie, but we can see that KNN is most accurate among them, and it is trying to predict movie which has very similar genre and if we try to google search those movie, google is also trying to give the similar recommendations.

5. Limitations

1. **Sparsity Problem**: Sparsity problem is one of the major problems encountered by recommender system and data sparsity has great influence on the quality of recommendation. The main reason behind data sparsity is that most users do not rate most of the items and the available ratings are usually sparse. Collaborative filtering suffers from this problem because it is dependent over the rating matrix in most cases.

2. **Cold start problem**: Cold start problem refers to the situation when a new user or item just enters the system. Three kinds of cold start problems are: new user problem, new item problem and new system problem. In such cases, it is really difficult to provide recommendation as in case of new user, there is very less information about user that is available and for a new item, no ratings are usually available and thus collaborative filtering can make useful recommendations in case of new item as well as new user. However, content-based methods can provide recommendation in case of new item as this method do not depends on any previous rating information of other users to recommend the item.

3. **Computation Resource and Time**: For KNN Algorithms we need computation power since this algorithm requires a lot of resource to run. Without enough hardware or resources, it is very difficult to run this algorithm. Also, this algorithm requires lot of computation time to execute. However, collaborative filtering and content-based filtering does require much time to run.

6. Conclusion

Several Recommendations system have been based on Content-Based, Collaborative-Based and Hybrid-Based Filtering and so far, most of them able to solve the problems while providing better recommender system.

After generating the recommendations out of inputs, businesses like OTT (Netflix,AmazonPrime,Hotstar,etc) and Websites like MovieLens, Flickmetrix, Popcornic,etc can apply preferred filters to adjust the final recommendations.

7. Reference

[1] https://www.researchgate.net/publication/340436258_Movie_recommendation_system_with_Collaborative_Filtering_using_K-NN

[2] https://le-james94.medium.com/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223

[3] https://iopscience.iop.org/article/10.1088/1742-6596/1964/4/042081

[4] https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea