UE20CS332: Algorithms for Web and Information Retrieval

ASSIGNMENT 2

TITLE: NETFLIX RECOMMMENDATION SYSTEM

TEAM MEMBERS:

PES2UG20CS528 NEERAJA N

PES2UG20CS539 RITHIKA A

PES2UG20CS541 S KUSHALA

PES2UG20CS544 SAKSHI DESHPANDE

SECTION 1: Problem statement

To create an accurate and effective recommendation system that can suggest movies and TV shows to users based on their viewing history, ratings, and other relevant user data. The recommendation system should be able to analyse large amounts of data and identify patterns in user behaviour to make personalized recommendations that are relevant and useful to each individual user which is done with the help of collaborative and content based filtering.

SECTION 2: Introduction

Netflix is a popular streaming platform that provides its users with access to a vast library of movies and TV shows. With so much content available, it can be overwhelming for users to find movies and TV shows that match their preferences. Hence we need a recommendation system.

The content-based approach focuses on the attributes of the movies and TV shows, such as title, cast, director, and description, to identify similarities and make recommendations based on the user's preferences.

The collaborative filtering approach analyses the user's viewing history and ratings to identify patterns and similarities with other users, and then recommends movies and TV shows based on those patterns.

Hybrid recommendation system combines these approaches to provide users with more accurate and relevant recommendations, increasing user engagement and satisfaction.

SECTION 3: Data set description

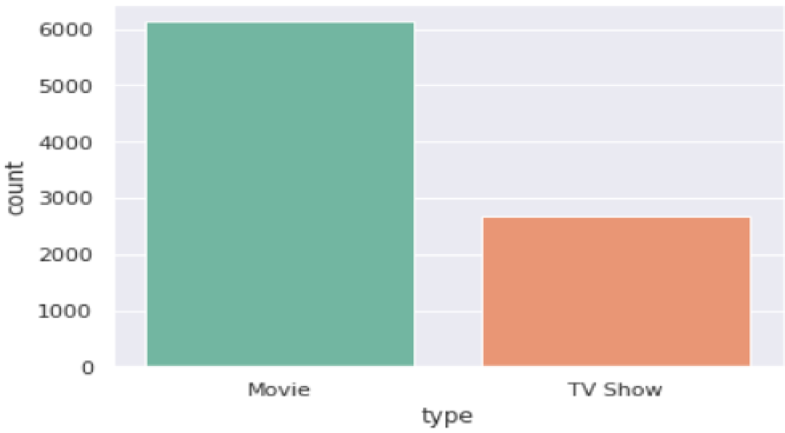
Dataset: https://www.kaggle.com/datasets/padmapriyatr/netflix-titles

The dataset consists of 8807 rows and 12 columns which includes show_id, type, director, cast, rating, description etc. The features from type, description, director, rating is considered for content based recommendation. Later the customer id and ratings(on a scale from 1 to 5) is added randomly to the movies and shows in the dataset to make collaborative(item based and user based) recommendation.

SECTION 4: EDA

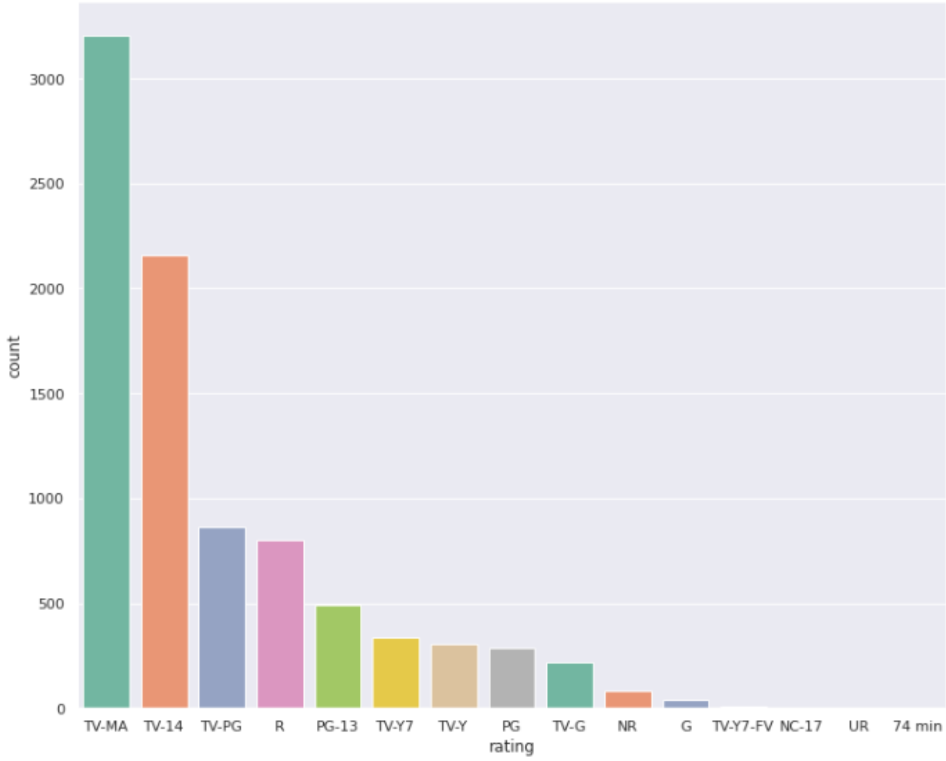
a. **Analysis of movies vs shows:**

```
sns.set(style="darkgrid")
ax = sns.countplot(x="type", data=df, palette="Set2")
```



b. Movie rating analysis

```
plt.figure(figsize=(12,10))
sns.set(style="darkgrid")
ax = sns.countplot(x="rating", data=df, palette="Set2", order=df['rating'].value_counts().index[0:15])
```



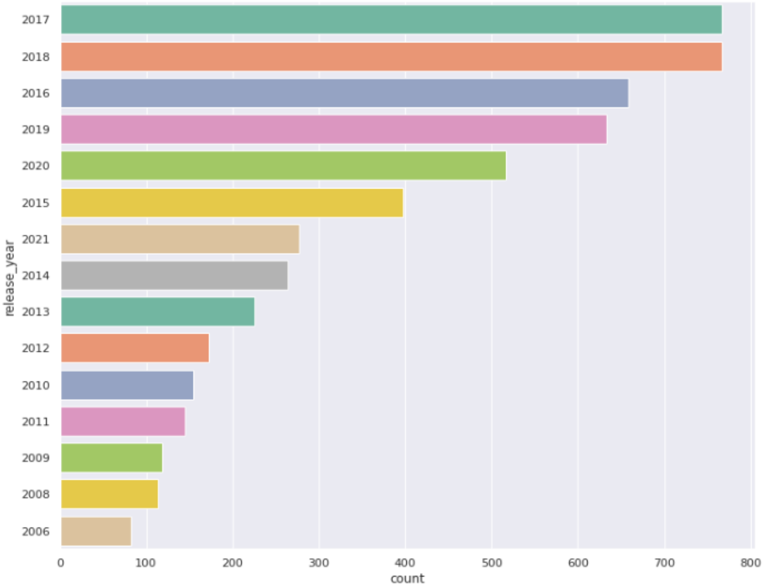
The largest count of movies are given 'TV-MA' rating that is a rating assigned by mature audiences only.

Second largest is the 'TV-14' stands for content that may be inappropriate for children younger than 14 years of age.

Third largest is TV-PG rating that stands for "Parental Guidance Suggested".

c. Year wise analysis for movies:

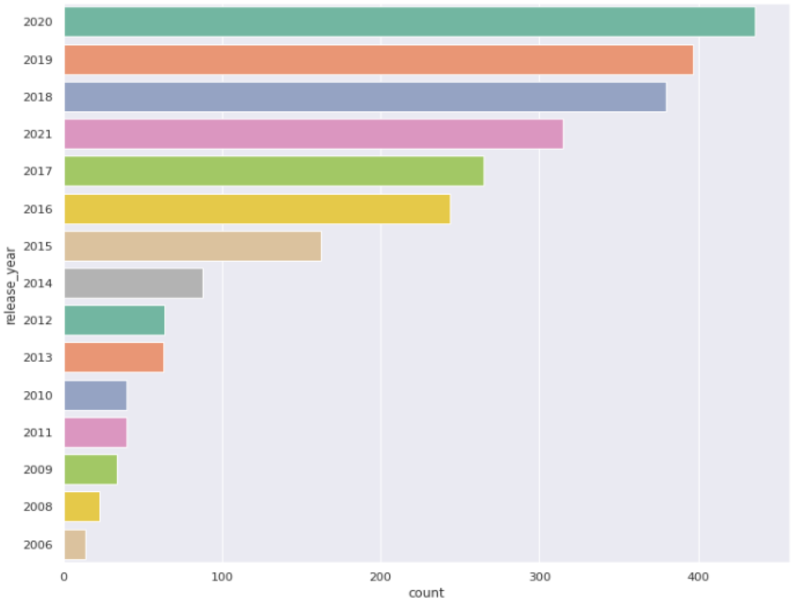
```
netflix_movies=df[df['type']=='Movie']
plt.figure(figsize=(12,10))
sns.set(style="darkgrid")
ax = sns.countplot(y="release_year", data=netflix_movies, palette="Set2", order=netflix_movies['release_year'].value_counts().index[0:15])
```



Most of the movies were released in the years 2017 and 2018.

d. Year wise analysis for shows:

```
netflix_shows=df[df['type']=='TV Show']
plt.figure(figsize=(12,10))
sns.set(style="darkgrid")
ax = sns.countplot(y="release_year", data=netflix_shows, palette="Set2", order=netflix_shows['release_year'].value_counts().index[0:15])
```



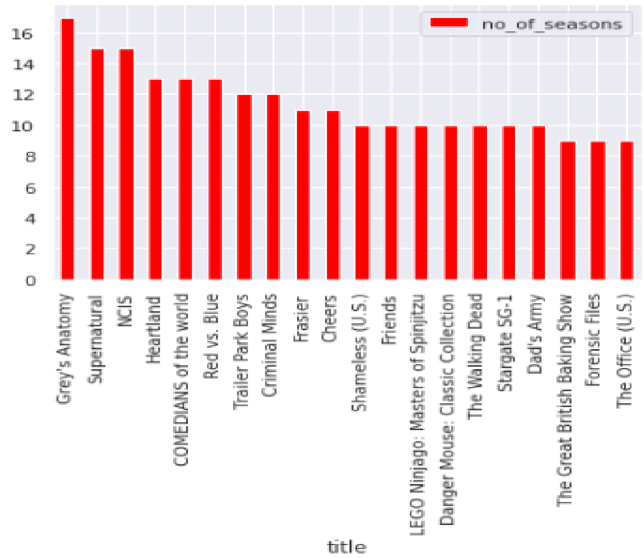
Most of the series were released in the year 2020.

e. **TV shows with largest number of seasons:**

```
t=['title','no_of_seasons']
top=durations[t]

top=top.sort_values(by='no_of_seasons', ascending=False)
top20=top[0:20]
top20.plot(kind='bar',x='title',y='no_of_seasons', color='red')
```

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



Grey's Anatomy has largest number of seasons.

SECTION 5: Preprocessing

a. Handling the missing values:

```
pd.DataFrame({'Total missing values':df.isna().sum(),
              'Percentage':(df.isna().sum()/len(df))*100})
```

	Total missing values	Percentage
show_id	0	0.000000
type	0	0.000000
title	0	0.000000
director	2634	29.908028
cast	825	9.367549
country	831	9.435676
date_added	10	0.113546
release_year	0	0.000000
rating	4	0.045418
duration	3	0.034064
listed_in	0	0.000000
description	0	0.000000

If you take a look at the missing values in this dataset, you will realize that the director column has 2634 NaN values which correspond with almost 30 percents of total data in that column. So, we can't just drop the NaN values because we will lose lots of movies to be given, instead we just fill the NaN values with empty string.

b. Reducing the data into bag of words which is used in content based filtering using tf-idf

```
new_df['type'] = new_df['type'].apply(remove_space)
new_df['director'] = new_df['director'].apply(separate)
new_df['cast'] = new_df['cast'].apply(separate)
new_df['rating'] = new_df['rating'].apply(remove_space)
new_df['listed_in'] = new_df['listed_in'].apply(separate)
new_df['description'] = new_df['description'].apply(remove_punc)

new_df.head()
```

	type	director	cast	rating	listed_in	description
title						
Dick Johnson Is Dead	movie	kirstenjohnson		pg-13	documentaries	as her father nears the end of his life filmma...
Blood & Water	tvshow		amaqamata khosingema gailmabalane thabangmolab...	tv-ma	internationaltvshows tvdramas tvmysteries	after crossing paths at a party a cape town te...
Ganglands	tvshow	julienleclercq	samibouajila tracygotoas samueljouy nabihaakka...	tv-ma	crimetvshows internationaltvshows tvaction&adv...	to protect his family from a powerful drug lor...
Jailbirds New Orleans	tvshow			tv-ma	docuseries realitytv	feuds flirtations and toilet talk go down amon...
Kota Factory	tvshow		mayurmore jitendrakumar ranjanraj alamkhan ahs...	tv-ma	internationaltvshows romantictvshows tvcomedies	in a city of coaching centers known to train i...

```
new_df['bag_of_words'] = ''

# Combine all the words into 1 column
for i, row in enumerate(new_df.iterrows()):
    string = ''
    for col in new_df.columns:
        if row[1][col] == '':
            continue
        else:
            string += row[1][col] + ' '
            new_df['bag_of_words'][i] = string.strip()

new_df.drop(new_df.columns[:-1], axis=1, inplace=True)
```

+ Code   + Markdown

```
new_df.head()
```

	bag_of_words
title	
Dick Johnson Is Dead	movie kirstenjohnson pg-13 documentaries as he...
Blood & Water	tvshow amaqamata khosingema gailmabalane thaba...
Ganglands	tvshow julienleclercq samibouajila tracygotoas...
Jailbirds New Orleans	tvshow tv-ma docuseries realitytv feuds flirta...
Kota Factory	tvshow mayurmore jitendrakumar ranjanraj alamk...

SECTION 6: Methodology (neighborhood based or model based collaborative filtering)

USER BASED USING SVD:

- Train the algorithm on the whole dataset using the fit method.
- We evaluate the algorithm using 5-fold cross-validation and print the results for the root mean squared error (RMSE) and mean absolute error (MAE).
- Once the model is trained, we can use it to make movie recommendations to users based on their ratings and viewing history.

```python
# import libraries
import pandas as pd
from surprise import Dataset
from surprise import Reader
from surprise import SVD
from surprise.model_selection import cross_validate


reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(data[['cust_id', 'show_id', 'ratingsnew']], reader)

# define the algorithm
algo = SVD()

# evaluate the algorithm using cross-validation
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

# train the algorithm on the whole dataset
trainset = data.build_full_trainset()
algo.fit(trainset)
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.3966  1.4539  1.4498  1.4157  1.4395  1.4311  0.0218
MAE (testset)     1.1917  1.2566  1.2554  1.2173  1.2460  1.2334  0.0252
Fit time          0.53    0.53    0.54    0.54    0.54    0.54    0.00
Test time         0.01    0.01    0.01    0.01    0.01    0.01    0.00
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7ddb2a003650>
```

RESULTS FOR USER-BASED:

```python
# make recommendations for a particular user
cust_id = 3
items_to_predict = df.loc[~df.show_id.isin(trainset.ur[cust_id]), 'show_id']
testset = [[cust_id, item_id, 4.] for item_id in items_to_predict]
predictions = algo.test(testset)

# sort the predicted ratings in descending order and get the top recommendations
top_n = 10
recommended_items = [pred.iid for pred in sorted(predictions, key=lambda x: x.est, reverse=True)][:top_n]
recommended_items
```

```
['s8579',
 's3901',
 's5959',
 's3055',
 's7961',
 's6010',
 's420',
 's560',
 's905',
 's161']
```

ITEM BASED USING PEARSON'S CORRELATION:

- The system identifies items that are similar to the ones the user has liked in the past, based on their attributes. Here it predicts the rating of the selected item.
- In sim_options set user_based to False to indicate that we are using item-based collaborative filtering.
- Finally, we sort the predicted ratings and select the top recommended users based on their predicted ratings.

```python
import pandas as pd
from surprise import Dataset
from surprise import Reader
from surprise import KNNBasic
from surprise.model_selection import train_test_split

# Load the data from a file (in this case, a CSV file)
df = pd.read_csv('/kaggle/working/new_dataset.csv')
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(df[['cust_id', 'show_id', 'ratingsnew']], reader)

# Split the data into a training set and a test set
trainset, testset = train_test_split(data, test_size=0.25)

# Use item-based collaborative filtering with KNNBasic algorithm
sim_options = {'name': 'cosine', 'user_based': False}
algo = KNNBasic(sim_options=sim_options)
algo.fit(trainset)
```

RESULTS FOR ITEM-BASED:

```
# Get the item ID from the user
item_id = "s3"

# Predict the rating for the selected item and print the result
item_inner_id = algo.trainset.to_inner_iid(item_id)
item_neighbors = algo.get_neighbors(item_inner_id, k=10)
item_neighbors = [algo.trainset.to_raw_iid(inner_id) for inner_id in item_neighbors]
item_ratings = []
for neighbor in item_neighbors:
    item_ratings.append(algo.predict(uid='user', iid=neighbor).est)
item_average_rating = sum(item_ratings) / len(item_ratings)
print("The predicted rating for item {} is {:.2f}.".format(item_id, item_average_rating))
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
The predicted rating for item s3 is 2.98.
```

SECTION 7: Content based recommendation

Using TF-IDF to represent the features of each item (e.g. movie or show) in the dataset, and recommend similar items to a user based on their preferences.

```
def recommendation(title, total_result=5, threshold=0.5):
    # Get the index
    idx = final_df[final_df['title'] == title].index[0]
    # Create a new column for similarity, the value is different for each title you input
    final_df['similarity'] = cosine_sim[idx]
    sort_final_df = final_df.sort_values(by='similarity', ascending=False)[1:total_result+1]

    # You can set a threshold if you want to norrow the result down
    #sort_final_df = sort_final_df[sort_final_df['similarity'] > threshold]

    # Is the title a movie or tv show?
    movies = sort_final_df['title'][sort_final_df['type'] == 'Movie']
    tv_shows = sort_final_df['title'][sort_final_df['type'] == 'TV Show']

    if len(movies) != 0:
        print('Similar Movie(s) list:')
        for i, movie in enumerate(movies):
            print('{}. {}'.format(i+1, movie))
        print()
    else:
        print('Similar Movie(s) list:')
        print('-\n')

    if len(tv_shows) != 0:
        print('Similar TV_show(s) list:')
        for i, tv_show in enumerate(tv_shows):
            print('{}. {}'.format(i+1, tv_show))
    else:
        print('Similar TV_show(s) list:')
        print('-')
```

SECTION 8: Results

```
recommendation('Breaking Bad')
```

```
Similar Movie(s) list:
1. The Show
2. The Book of Sun

Similar TV_show(s) list:
1. Better Call Saul
2. Marvel's The Punisher
3. Dare Me
```

+ Code + Markdown

```
recommendation('Narcos')
```

```
Similar Movie(s) list:
-

Similar TV_show(s) list:
1. Narcos: Mexico
2. Wild District
3. El Cartel
4. Miss Dynamite
5. Cocaine Cowboys: The Kings of Miami
```

```
recommendation('Chappie')
```

```
Similar Movie(s) list:
1. Real Steel
2. District 9
3. 2036 Origin Unknown
4. Singularity
5. AlphaGo

Similar TV_show(s) list:
-
```

+ Code + Markdown

```
recommendation('Ganglands')
```

```
Similar Movie(s) list:
1. Earth and Blood
2. Chhota Bheem: The Rise of Kirmada
3. Paradise Beach
4. Bright

Similar TV_show(s) list:
1. The Eagle of El-Se'eed
```

SECTION 9: Hybrid model (combined model of content and collaborative filtering)

- The content-based approach(KNN) is used to compute the cosine similarity matrix using description of shows.
- The collaborative filtering approach is used to create a matrix factorization model using Singular Value Decomposition (SVD) to predict a user's show ratings based on the ratings of other similar users.
- The output of this code is a list of recommended shows that combines both content-based and collaborative filtering approaches using a weighted average.

```
# Generate recommendations
def hybrid(cust_id, show_title):
    # Get index of movie
    idx = df.loc[df['title'] == show_title].index[0]

    # Calculate cosine similarity scores
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort shows based on similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:31]

    # Get show indices
    show_indices = [i[0] for i in sim_scores]

    # Get show titles and similarity scores
    shows = df.iloc[show_indices][['show_id', 'title', 'type']]
    shows['similarity'] = [cosine_sim[idx][i] for i in show_indices]

    # Predict ratings using SVD algorithm
    shows['est'] = shows['show_id'].apply(lambda x: algo.predict(cust_id, x).est)

    # Predict ratings using KNN algorithm
    shows['est2'] = shows['show_id'].apply(lambda x: knn.predict(cust_id, x).est)

    # Calculate final ratings as a weighted average of the two predicted ratings
    shows['final_rating'] = 0.6 * shows['est'] + 0.4 * shows['est2']

    # Sort shows based on final ratings
    shows = shows.sort_values('final_rating', ascending=False)

    # Return top 10 recommended shows
    return shows.head(10)
```

RESULTS OF HYBRID MODEL:

```python
# Generate recommendations for user with ID 3 and movie "Ganglands"
print(hybrid(3, 'Ganglands'))
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
     show_id                        title     type  similarity       est  \
1349   s1350  Pablo Escobar, el patrón del mal  TV Show    0.110105  3.430174
5305   s5306                           Narcos  TV Show    0.150355  3.273710
1905   s1906                     Cold Harbour    Movie    0.135985  3.150534
3297   s3298                   Paradise Beach    Movie    0.121522  3.147350
2549   s2550                       John Henry    Movie    0.112155  3.146716
8569   s8570               The Young Vagabond    Movie    0.121704  3.055689
711     s712                         Security    Movie    0.147345  3.544718
42       s43                           Jaws 2    Movie    0.102069  3.383280
5113   s5114                           Bright    Movie    0.163819  3.283077
3976   s3977            The Eagle of El-Se'eed  TV Show    0.128196  3.097691

      est2  final_rating
1349   5.0      4.058104
5305   5.0      3.964226
1905   5.0      3.890320
3297   5.0      3.888410
2549   5.0      3.888030
8569   5.0      3.833414
711    4.0      3.726831
42     4.0      3.629968
5113   4.0      3.569846
3976   4.0      3.458614
```

SECTION 10: Conclusion

- For content based filtering, **Earth and blood** is the top recommendation
- For collaborative filtering, **Blood and water** is the top recommendation
- For hybrid approach, **Moshe Kasher: Live in Oakland** is the top recommendation