PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

Report on

"Recommender Engines"

Submitted by

AAKANKSH GOWDA (PES1UG20EC004)
ANIRUDHAN R (PES1UG20EC029)
AYUSH ANAND (PES1UG20EC043)

For the course

UE20EC352 – Machine Learning And Applications

Jan – May 2023

under the guidance of

Dr. Manikandan J
Department of ECE
PES University

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

PROGRAM - B.TECH

TABLE OF CONTENTS

Motivation for the Domain and the Problem	3
Content Based Recommendation Engine	5
Dataset Description	5
Algorithm	6
Code	7
Result	9
Observation	10
User Based Recommendation Engine	10
Dataset Description	10
Algorithm	11
Code	11
Result	13
Observation	14
Genre Based Recommendation Engine	15
Dataset Description	15
Algorithm	15
Code	15
Result	17
Observation	17

Motivation for the Domain and the Problem:

- The motivation of this problem statement is to help all those who do not know their interests to identify and share the same with the anime community.

  Personalization: Recommender systems can provide personalized recommendations to users based on their individual preferences and behavior. This can help users discover new products or services that they may not have otherwise found on their own.

  Increased engagement: Anime is a huge community but many people still struggle to find out where their interest lies. By providing personalized recommendations, recommender systems can increase user engagement with a product or service. This can lead to increased user satisfaction and loyalty.

  Improved customer experience: Recommender systems can help users find what they are looking for more quickly and easily, improving the overall customer experience.

  Data collection: Recommender systems can collect data on user behavior and preferences, which can be used to improve the system over time and inform other decisions.

- A recommender system could help people who are struggling to find their interest in anime by providing personalized recommendations based on their viewing history and preferences.

- Recommender systems are algorithms that use data to suggest items to users based on their behavior, preferences, and past interactions. In the context of anime, a recommender system could analyze a user's viewing history, ratings, and comments on anime they have watched, and use that data to recommend new anime titles that are likely to appeal to the user.

- For example, if a user has watched several action anime series and given them high ratings, the recommender system could suggest other action anime series that are similar in genre or style. If the user has expressed a preference for anime with strong character development, the recommender system could suggest titles with well-developed characters.

- By using a recommender system, users can discover new anime titles that they may not have otherwise found on their own, and they can also save time and effort in searching for anime that aligns with their interests.

- Overall, a recommender system can be a valuable tool for helping people discover and explore their interests in anime.

- Recommender engines are basically going to understand either the user persona or the content given to it

- Recommender engines are going to give us outputs of what to watch next by checking the shows that are most closely related to the previous shows one has watched.

- So, even if recommender engines are its own type of problem, it can broadly be considered as a regression problem.

- We have shown 3 types of recommender engines
    1. Content Based Recommendation
    2. Collaborative Filtering / User based Recommendation
    3. Genre Based Recommendation

Content Based Recommendation Engine

Dataset Description

For the content-based model, we have used 3 different datasets

- Score Dataset
- Anime Dataset
- User Dataset

anime_cleaned		users_cleaned		animelists_cleaned	
Features	33	Features	17	Features	11
Classes	none	Classes	none	Classes	none
Regression		Regression		Regression	
Training samples	5001	Training samples	81534	Training samples	786432
Testing samples	User Input	Testing samples	User Input	Testing samples	User Input
Datapoints	6668	Datapoints	108711	Datapoints	1048576

Score Dataset:

- This has 11 features, out of which we are going to be using 4 features for the recommender. These are 'username', 'anime_id', 'my_score', 'my_status'
- These features are used to get ratings of different shows from different users and also their watching status.
- There are a total of around 10 lakh data points which consist of anime ID and its ratings

Anime Dataset:

- This has 33 features, out of which we are going to be using 4 features for the recommender. These features are ‘anime_id’, ‘title’, ‘genre’, ‘type’
- These features are are the ones that is going to be describing any and every show in the dataset
- There are a total of around 6000 data points which holds information about different shows.

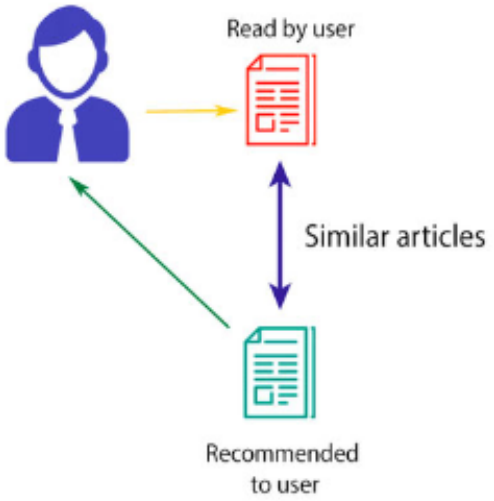
User Dataset:

- This has 17 features out of which, we are going to be using 2 features. These features are ‘username’ and ‘user_id’
- These features are the ones that is going to be giving us information about the users so that the scores and the anime dataset can be pivoted together
- There are a total of 1 Lakh data points, which consists of the user ID and usernames

Algorithm

A Content-Based recommender engine is a type of recommender system that recommends items based on their features or attributes.

The engine creates a profile for each user based on their past interactions and recommends items that have similar features or attributes to those the user has previously interacted with.



The Algorithm used in the content based recommender engine is the Baseline algorithm

Baseline algorithms aims to predict the rating of an item based on its features and the overall user-item rating distribution

The baseline estimate can then be used as a starting point for making more personalized recommendations to users.

$$b_{ui} = \mu + b_u + b_i$$

b_{ui} = Baseline estimate rating of item i by user u
μ = Overall mean rating across all items and users
b_u = User bias
b_i = Item bias

Code

```
from collections import defaultdict
import pandas as pd
import numpy as np
import scipy
from scipy.sparse.linalg import svds
import matplotlib.pyplot as plt
import surprise as sp
import time
UsersDF = pd.read_csv('/content/users_cleaned.csv')
AnimesDF = pd.read_csv('/content/anime_cleaned.csv')
ScoresDF = pd.read_csv('/content/animelists_cleaned_new.csv')
print(AnimesDF.shape)
AnimesDF.head()
UsersDF.head()
ScoresDF.head()
ScoresDF = ScoresDF[['username', 'anime_id', 'my_score', 'my_status']]
ScoresDF['my_score'].describe().apply(lambda x: format(x, '.2f')).reset_index()
lower_rating = ScoresDF['my_score'].min()
upper_rating = ScoresDF['my_score'].max()
print('Range of ratings vary between: {0} to {1}'.format(lower_rating, upper_rating))
UsersAndScores = ScoresDF['username'].value_counts().reset_index().rename(columns={"username": "animes_rated", "index":
"username"})
UsersSampled = UsersDF.sample(frac = .01, random_state = 2)
UsersSampled.head()
UsersAndScoresSampled = pd.merge(UsersAndScores, UsersSampled, left_on = 'username', right_on = 'username', how =
'inner')
UserRatedsAggregated =
UsersAndScoresSampled['animes_rated'].value_counts().reset_index().rename(columns={"animes_rated": "group_size",
"index": "animes_rated"}).sort_values(by=['animes_rated'])
RatedsPerAnime = ScoresDF['anime_id'].value_counts().reset_index().rename(columns={"anime_id": "number_of_users",
"index": "anime_id"})
RatedsPerAnime.head()
AnimeRatedsAggregated =
RatedsPerAnime['number_of_users'].value_counts().reset_index().rename(columns={"number_of_users": "group_size", "index":
"number_of_users"}).sort_values(by=['number_of_users'])
AnimeRatedsAggregated.head(n = 30)
UserRatedsCutten = UsersAndScoresSampled[UsersAndScoresSampled['animes_rated'] >= 10]
AnimeRatedsCutten = RatedsPerAnime[RatedsPerAnime['number_of_users'] >= 10]

ScoresDFHotStart = pd.merge(ScoresDF, UserRatedsCutten, left_on = 'username', right_on = 'username', how = 'inner')
ScoresDFHotStart = pd.merge(ScoresDFHotStart, AnimeRatedsCutten, left_on = 'anime_id', right_on = 'anime_id', how =
'inner')
print('The initial dataframe has {0} registers and the sampled one has {1} rows.'.format(ScoresDF['username'].count(),
ScoresDFHotStart['username'].count()))
def precision_recall_at_k(predictions, k=10, threshold= 7):
    '''Return precision and recall at k metrics for each user.'''
    user_est_true = defaultdict(list)
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))
    precisions = dict()
    recalls = dict()
    for uid, user_ratings in user_est_true.items():
        user_ratings.sort(key=lambda x: x[0], reverse=True)
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)
```

```
        n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])
        n_rel_and_rec_k = sum(((true_r >= threshold) and (est >= threshold))
                              for (est, true_r) in user_ratings[:k])
        precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 1
        recalls[uid] = n_rel_and_rec_k /  n_rel if n_rel != 0 else 1
    print('Precision = ', precisions, 'recall = ', recalls)
    return precisions, recalls
random_state = 42
reader = sp.Reader(rating_scale=(0, 10))
data = sp.Dataset.load_from_df(ScoresDFHotStart[['username', 'anime_id', 'my_score']], reader)
trainset, testset = sp.model_selection.train_test_split(data, test_size=.25, random_state = random_state)
analysis = defaultdict(list)
test_dict = {'SVD' : sp.SVD(random_state=random_state), 'SlopeOne' : sp.SlopeOne(), 'NMF' :
sp.NMF(random_state=random_state), 'NormalPredictor' : sp.NormalPredictor(), 'KNNBaseline' :
sp.KNNBaseline(random_state=random_state), 'KNNBasic' : sp.KNNBasic(random_state=random_state), 'KNNWithMeans' :
sp.KNNWithMeans(random_state=random_state), 'KNNWithZScore' : sp.KNNWithZScore(random_state=random_state),
'BaselineOnly' : sp.BaselineOnly(), 'CoClustering': sp.CoClustering(random_state=random_state)}
for key, value in test_dict.items():
    start = time.time()
    value.fit(trainset)
    predictions = value.test(testset)
    rmse = sp.accuracy.rmse(predictions)
    precisions, recalls = precision_recall_at_k(predictions, k=10, threshold=7)
    precision_avg = sum(prec for prec in precisions.values()) / len(precisions)
    analysis[value] = (key, rmse, precision_avg, time.time() - start)
print(analysis)
analysis_df = pd.DataFrame.from_dict(analysis, orient = 'index', columns = ['Algorithm', 'RMSE', 'Precision@10', 'Time to run
(in seconds)']).reset_index()
analysis_df = analysis_df[['Algorithm', 'RMSE', 'Precision@10', 'Time to run (in seconds)']]
analysis_df = analysis_df.sort_values(by=['Precision@10'], ascending = False)
analysis_df['RMSE^-1'] = analysis_df['RMSE'] ** -1
analysis_df.head(n = 15)
analysis_df.plot(x = 'Algorithm', y = 'RMSE', kind = 'bar')
plt.show()
print('\n')
analysis_df.plot(x = 'Algorithm', y = 'Precision@10', kind = 'bar')
plt.show()
print('\n')
analysis_df.plot(x = 'Algorithm', y = 'Time to run (in seconds)', kind = 'bar')
plt.show()
print('\n')
als_param_grid = {'bsl_options': {'method': ['als'],
                          'reg_i': [5, 10, 15],
                          'reg_u': [10, 15, 20],
                          'n_epochs': [5, 10, 15, 20]
                          }
            }

sgd_param_grid = {'bsl_options': {'method': ['sgd'],
                          'reg': [0.01, 0.02, 0.03],
                          'n_epochs': [5, 10, 15, 20],
                          'learning_rate' : [0.001, 0.005, 0.01]
                          }
            }

als_gs = sp.model_selection.GridSearchCV(sp.BaselineOnly, als_param_grid, measures=['rmse'], cv = 3, joblib_verbose = 0)
sgd_gs = sp.model_selection.GridSearchCV(sp.BaselineOnly, sgd_param_grid, measures=['rmse'], cv = 3, joblib_verbose = 0)
als_gs.fit(data)
```

```
sgd_gs.fit(data)
print(sgd_gs.best_score['rmse'])
print(sgd_gs.best_params['rmse'])
trainset = data.build_full_trainset()
algo = sp.BaselineOnly()
algo.fit(trainset)
testset = trainset.build_anti_testset()
predictions = algo.test(testset)
last_predictions = pd.DataFrame(predictions, columns=['uid', 'iid', 'rui', 'est', 'details'])
last_predictions.drop('rui', inplace = True, axis = 1)
def bringing_first_n_values(df, uid, n=10):
    df = df[df['uid'] == uid].nlargest(n, 'est')[['uid', 'iid', 'est']]
    df = pd.merge(df, AnimesDF, left_on = 'iid', right_on = 'anime_id', how = 'left')
    return df[['uid', 'est', 'title', 'genre']]
bringing_first_n_values(last_predictions, 'Tomoki-sama')
sim_options = {'name': 'pearson_baseline', 'user_based': False}
algo_items = sp.KNNBaseline(sim_options=sim_options)
algo_items.fit(trainset)
def get_item_recommendations(anime_title, anime_id=3000000, k=10):
    if anime_id == 3000000:
        anime_id = AnimesDF[AnimesDF['title'] == anime_title]['anime_id'].iloc[0]
    iid = algo_items.trainset.to_inner_iid(anime_id)
    neighbors = algo_items.get_neighbors(iid, k=k)
    raw_neighbors = (algo.trainset.to_raw_iid(inner_id) for inner_id in neighbors)
    df = pd.DataFrame(raw_neighbors, columns = ['Anime_ID'])
    df = pd.merge(df, AnimesDF, left_on = 'Anime_ID', right_on = 'anime_id', how = 'left')
    return df[['Anime_ID', 'title', 'genre']]
get_item_recommendations('Kimi no Na wa.', k=5)
```

Result

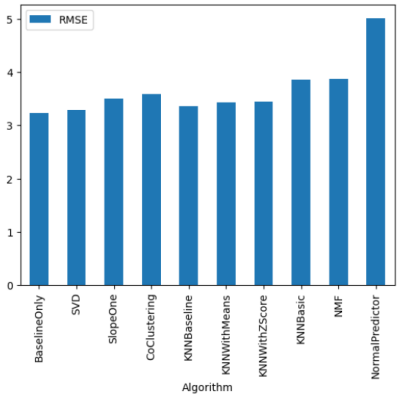
```
RMSE: 3.2955
RMSE: 3.5025
RMSE: 3.8693
RMSE: 4.9218
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 3.3609
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 3.8629
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 3.4338
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 3.4415
Estimating biases using als...
RMSE: 3.2290
RMSE: 3.5893
```

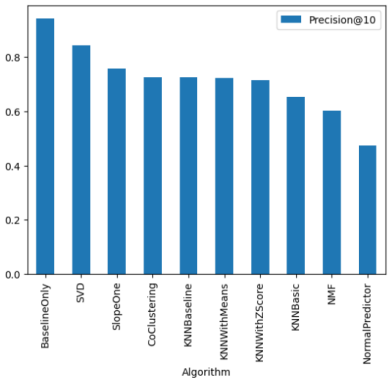
```
get_item_recommendations('Toradora!', k=5)
```

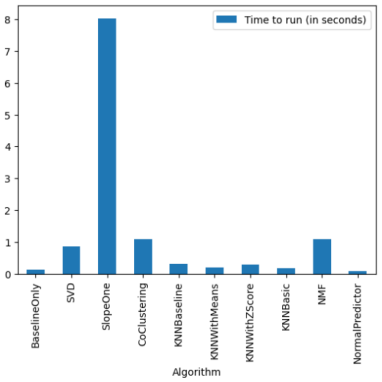
	Anime_ID	title	genre
0	15583	Date A Live	Comedy, Harem, Mecha, Romance, School, Sci-Fi
1	7054	Kaichou wa Maid-sama!	Comedy, Romance, School, Shoujo
2	15225	Hentai Ouji to Warawanai Neko.	Harem, Comedy, Supernatural, Romance, School
3	7791	K-On!!	Comedy, Music, School, Slice of Life
4	8769	Ore no Imouto ga Konnani Kawaii Wake ga Nai	Slice of Life, Comedy

Observation

We have compared some algorithms - Baseline, SVD, SlopeOne, CoClustering, KNNBaseline, KNNWithMeans, KNNWithZScore, KNNBasic, NMF and Normal Predictor. It is seen that the RSME is the lowest and the precision is the highest for the baseline algorithm. This is why we proceeded with this algorithm







User Based Recommender Engine

Dataset Description

For the User based model, we have used 2 different datasets, namely

- Anime dataset
- Rating dataset

These 2 datasets are pivoted to make 1 global dataset on which the model runs

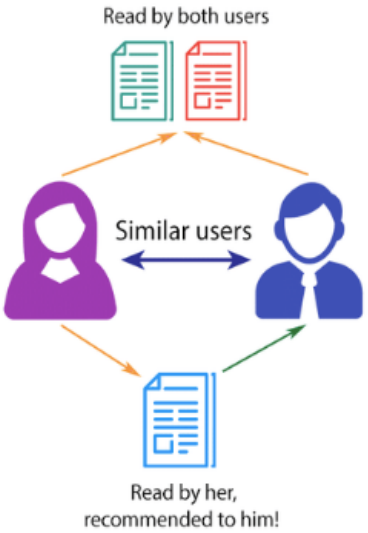
anime		rating		Pivoted	
Features	7	Features	3	Features	9
Classes	none	Classes	none	Classes	none
Regression		Regression		Regression	
Training samples	9869	Training samples	7813737	Training samples	7813610
Testing samples	–	Testing samples	–	Testing samples	100
Datapoints	12294	Datapoints	108711	Datapoints	1048576

- The pivoted dataset has 9 features, out of which we have used 5 features. These features are ‘anime_id’, ‘name’, ‘genre’, ‘rating’, ‘user_id’, ‘user_rating’

- Since this is a pivoted database, it will have the information about each show and with that there will be the information about each user and their ratings along with their user ID

Algorithms

The goal of a user-based recommender engine is to recommend items that are relevant and interesting to the user based on the preferences and behavior of similar users
This engine creates a profile for each user based on their past interactions and recommends items that have been rated highly by users who have similar profiles.



For the User Based recommender engine, a vector space model is employed.

Each item is stored as a vector of its attributes in an n-dimensional space and the angles between the vectors are calculated to determine the similarity between the vectors.

The user profile vectors are also created based on his actions on previous attributes of items and the similarity between an item and a user is determined.

Relevance is compared by the deviation angle between the input and each vector, and this is done by the cosine similarity i.e. the cosine of the angle is considered as the deviation.

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
anime = pd.read_csv("/content/anime.csv")
```

```
rating = pd.read_csv("/content/rating.csv")
print(f"Shape of The Anime Dataset : {anime.shape}")
print(f"\nGlimpse of The Dataset :")
anime.head()
print(f"Informations About Anime Dataset :\n")
print(anime.info())
print(f"Shape of The Rating Dataset : {rating.shape}")
print(f"\nGlimpse of The Dataset :")
rating.head()
print(f"Informations About Rating Dataset :\n")
print(rating.info())
print("Null Values of Anime Dataset :")
anime.isna().sum().to_frame().T.style
print("After Dropping, Null Values of Anime Dataset :")
anime.dropna(axis = 0, inplace = True)
anime.isna().sum().to_frame().T.style
dup_anime = anime[anime.duplicated()].shape[0]
print(f"There are {dup_anime} duplicate entries among {anime.shape[0]} entries in anime dataset.")
print(f"Summary of The Rating Dataset :")
rating.describe().T.style
print("Null Values of Rating Dataset :")
rating.isna().sum().to_frame().T.style
dup_rating = rating[rating.duplicated()].shape[0]
print(f"There are {dup_rating} duplicate entries among {rating.shape[0]} entries in rating dataset.")
rating.drop_duplicates(keep='first',inplace=True)
print(f"\nAfter removing duplicate entries there are {rating.shape[0]} entries in this dataset.")
fulldata = pd.merge(anime,rating,on="anime_id",suffixes= [None, "_user"])
fulldata = fulldata.rename(columns={"rating_user": "user_rating"})
print(f"Shape of The Merged Dataset : {fulldata.shape}")
print(f"\nGlimpse of The Merged Dataset :")
fulldata.head()
data = fulldata.copy()
data["user_rating"].replace(to_replace = -1 , value = np.nan ,inplace=True)
data = data.dropna(axis = 0)
print("Null values after final pre-processing :")
data.isna().sum().to_frame().T.style
selected_users = data["user_id"].value_counts()
data = data[data["user_id"].isin(selected_users[selected_users >= 50].index)]
data_pivot_temp = data.pivot_table(index="name",columns="user_id",values="user_rating").fillna(0)
data_pivot_temp.head()
import re
def text_cleaning(text):
    text = re.sub(r'&amp;quot;', '', text)
    text = re.sub(r'.hack//', '', text)
    text = re.sub(r'&amp;#039;', '', text)
    text = re.sub(r'A&amp;#039;s', '', text)
    text = re.sub(r'I&amp;#039;', 'I\'', text)
    text = re.sub(r'&amp;amp;', 'and', text)
    return text
data["name"] = data["name"].apply(text_cleaning)
data_pivot = data.pivot_table(index="name",columns="user_id",values="user_rating").fillna(0)
print("After Cleaning the animes names, let's see how it looks like.")
data_pivot.head()
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import KNeighborsClassifier
data_matrix = csr_matrix(data_pivot.values)
print(data_matrix.shape)
```

```
model_knn = NearestNeighbors(metric = "cosine", algorithm = "brute")
model_knn.fit(data_matrix)
query_no = np.random.choice(data_pivot.shape[0]) # random anime title and finding recommendation
print(f"We will find recommendation for {query_no} no anime which is {data_pivot.index[query_no]}.")
distances, indices = model_knn.kneighbors(data_pivot.iloc[query_no,:].values.reshape(1, -1), n_neighbors = 11)
print(distances)
no = []
name = []
distance = []
rating = []
for i in range(0, len(distances.flatten())):
    if i == 0:
        print(f"Recommendations for {data_pivot.index[query_no]} viewers :\n")
    else:
        no.append(i)
        name.append(data_pivot.index[indices.flatten()[i]])
        distance.append(distances.flatten()[i])
        rating.append(anime[anime["name"]==data_pivot.index[indices.flatten()[i]]]["rating"].values)
dic = {"No" : no, "Anime Name" : name, "Rating" : rating}
recommendation = pd.DataFrame(data = dic)
recommendation.set_index("No", inplace = True)
recommendation
```

Result

```
Shape of The Merged Dataset : (7813610, 9)

Glimpse of The Merged Dataset :
```

	anime_id	name	genre	type	episodes	rating	members	user_id	user_rating
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	99	5
1	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	152	10
2	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	244	10
3	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	271	10
4	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630	278	-1

```
(9869, 32967)
Recommendations_for_Shinmai Maou no Testament_viewers

                        Anime Name  Rating
No
1   Shinmai Maou no Testament Burst  [7.13]
2                     Absolute Duo  [6.78]
3                     Trinity Seven  [7.43]
4      Seiken Tsukai no World Break  [7.16]
5             High School DxD BorN  [7.71]
6           Gakusen Toshi Asterisk  [7.14]
7      Seirei Tsukai no Blade Dance  [7.15]
8         Rakudai Kishi no Cavalry  [7.78]
9               Grisaia no Kajitsu  [7.67]
10             High School DxD New  [7.87]



Recommendations_for_Yatterman_viewers

                                      Anime Name  Rating
No
1    Super Mario World: Mario to Yoshi no Bouken Land  [6.02]
2                                 Yatterman Specials  [6.48]
3                                Yatterman x Toshiba  [4.32]
4   Wakusei Robo Danguard Ace tai Konchuu Robot Gu...  [6.08]
5   Dr. Slump: Hoyoyo! Arale no Himitsu Dai Koukai...  [6.17]
6                    Satto Anshin, SAT x Peeping Life  [4.07]
7                                 Umigame to Shounen  [6.55]
8       Konpeki no Kantai: Sourai Kaihatsu Monogatari  [6.45]
9                             Good Morning!!! Doronjo   [6.0]
10             Eguchi Hisashi no Kotobuki Gorou Show   [6.0]
```

Observation

For different anime ID generated for testing or if an anime input is given, it is seen that with respect to the user persona of people who liked similar shows, the vector distances of the next 10 closest distances are found and those shows are recommended to the user.

```
[[0.         0.49772836 0.52276588 0.64874601 0.66213131 0.66213131
  0.66213131 0.66213131 0.66213131 0.66213131 0.66213131]]
Recommendations_for_Prince of Tennis: Message in a Bottle_viewers

                                Anime Name  Rating
No
1                            Lupin Shanshei   [5.9]
2   Minna Atsumare! Falcom Gakuen SC Special  [5.57]
3                      Ketsuekigata-kun! 4  [6.55]
4              Jibun no Mune ni te wo Atete  [4.88]
5                             Furiten-kun  [6.56]
6                                   Spacy  [5.14]
7                             Dragon Fist  [5.29]
8                          Momoiro no Crayon  [5.06]
9                            Moshidora Recap  [5.94]
10                             Straw Byururu   [5.0]



[[0.         0.16555849 0.58454014 0.69064432 0.69064432 0.69457962
  0.69534707 0.70997905 0.71467167 0.74309409 0.77520733]]
Recommendations_for_Mini Hama: Minimum Hamatora_viewers

                                      Anime Name  Rating
No
1                Mini Hama: Minimum Hamatora Movies  [6.35]
2                     Nijiiro Days: Houkago Special  [6.54]
3       Q Transformers: Kaette Kita Convoy no Nazo  [5.72]
4                                 Pankis! 2-jigen  [6.08]
5                                     Fw:Hamatora   [7.2]
6                                        Aoi Heya  [4.77]
7        Utawarerumono: Itsuwari no Kamen Specials  [6.61]
8         Minna Atsumare! Falcom Gakuen SC Special  [5.57]
9   Q Transformers: Saranaru Ninki Mono e no Michi  [6.15]
10                                 The Green Wind  [5.06]
```

Genre Based Recommender Engine

Dataset Description

For the User based model, we have used 1 dataset, the anime dataset. This is because, we are going to be recommending shows only with respect to the genre and rating of that show

anime	
Features	7
Classes	none
Regression	
Training samples	9869
Testing samples	100
Datapoints	12294

- The anime dataset has 7 features, out of which we have used 4 features. These features are ‘anime id’, ‘name’, ‘genre’, ‘rating’
- Since this is a genre based recommender engine, we only care about the genre and the rating of that show. Therefore, these features are considered

Algorithm

For the Genre Based recommender engine, we have used Jaccard similarity.
Jaccard Similarity is a proximity measurement used to calculate the proximity between the genre of the input show and the genre of the recommended shows.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Code

```
import random
import csv
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import itertools
import collections
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import jaccard_score
animes = pd.read_csv('/content/anime.csv') # load the data
animes['genre'] = animes['genre'].fillna('None') # filling 'empty' data
animes['genre'] = animes['genre'].apply(lambda x: x.split(', ')) # split genre into list of individual genre
genre_data = itertools.chain(*animes['genre'].values.tolist()) # flatten the list
genre_counter = collections.Counter(genre_data)
genres = pd.DataFrame.from_dict(genre_counter,
orient='index').reset_index().rename(columns={'index':'genre', 0:'count'})
genres.sort_values('count', ascending=False, inplace=True)
# Plot genre
f, ax = plt.subplots(figsize=(8, 12))
sns.set_color_codes("pastel")
sns.set_style("white")
sns.barplot(x="count", y="genre", data=genres, color='b')
ax.set(ylabel='Genre',xlabel="Anime Count")
genre_map = {genre: idx for idx, genre in enumerate(genre_counter.keys())}
def extract_feature(genre):
    feature = np.zeros(len(genre_map.keys()), dtype=int)
    feature[[genre_map[idx] for idx in genre]] += 1
    return feature
anime_feature = pd.concat([animes['name'], animes['genre']], axis=1)
anime_feature['genre'] = anime_feature['genre'].apply(lambda x: extract_feature(x))
print(anime_feature.head(80))
animeID = []
for i in range(5):
    animeID.append(random.randint(1, 12294))
print(animeID)
test_data = anime_feature.take([animeID])
for row in test_data.iterrows():
    print('Similar anime like {}:'.format(row[1]['name']))
    search = anime_feature.drop([row[0]]) # drop current anime
    search['result'] = search['genre'].apply(lambda x: jaccard_score(row[1]['genre'], x))
    search_result = search.sort_values('result', ascending=False)['name'].head(5)
    for res in search_result.values:
        print('\t{}'.format(res))
    print()
```

Results

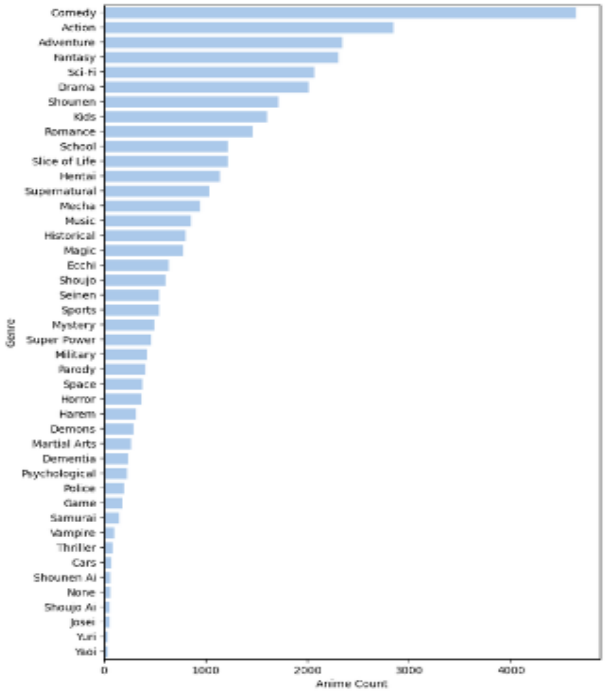
```
Similar anime like .hack//The Movie: Sekai no Mukou ni:
        .hack//Intermezzo
        .hack//Sign
        .hack//Quantum
        Rokka no Yuusha
        Rokka no Yuusha Picture Drama

Similar anime like Poppoya-san: Nonki Kikanshi:
        Kiki to Lala no Aoi Tori
        Rainbow no Anime Bako
        Shiawase no Ouji
        Shibawanko no Wa no Kokoro
        Kiki to Lala no Hansel to Gretel

Similar anime like Cossette no Shouzou:
        Elfen Lied
        Night Head Genesis
        Litchi DE Hikari Club
        Elfen Lied Special
        Umi Monogatari: Anata ga Ite Kureta Koto

Similar anime like Aura Battler Dunbine Memorial:
        Giant Gorg
        Genei Toushi Bastof Lemon
        Casshern: Robot Hunter
        Jikuu Boukenki Zentrix
        Chogattai Majutsu Robot Ginguiser Specials

Similar anime like Persona 3 the Movie Meets "Walkman":
        Melody
        Minna de Bokumetsu Inshu Unten
        Black Jack ONA
        Wake Up, Girls! no Miyagi PR Yarasete Kudasai!
        Nobara
```



Observation

From the produced results, it is seen that according to the genre of the input anime ID, using the jaccard score, the top 5 shows are recommended.

There are some recommendations with Jaccard score 1. This is due to the fact that if all the genres of two shows are matching, according to the Jaccard similarity formula, it is completely similar.

```
Similar anime like Hyouga Senshi Gaislugger:
90      1.0
294     1.0
394     1.0
546     1.0
1029    1.0
Name: result, dtype: float64
        Buddy Complex
        Hwanggeum Nalgae 1.2.3.
        Tokusou Kihei Dorvack
        Promised Town
        Run=Dim

Similar anime like Xiong Chu Mo Zhi Chunri Dui Dui Peng:
1378    1.0
3314    1.0
3959    1.0
4095    1.0
4473    1.0
Name: result, dtype: float64
        Xi Yang Yang Yu Hui Tai Lang: Zhi Hu Hu Sheng Wei
        Hazedon
        Mitsubachi Maya no Bouken
        Tottoko Hamtarou Movie 4: Hamtaro to Fushigi no Oni no Emon Tou
        Tottoko Hamtarou OVA 3: Hamuchanzu to Niji no Kuni no Oujisama - Sekai de Ichiban no Takaramono

Similar anime like High School DxD OVA:
1749    1.000000
1057    0.833333
2045    0.833333
3337    0.800000
3482    0.800000
Name: result, dtype: float64
        High School DxD New: Oppai, Tsutsumimasu!
        High School DxD
        High School DxD Specials
        Tanin no Kankei
        Sprite: Between Two Worlds

Similar anime like Noragami Aragoto OVA:
698     1.0
1887    1.0
2319    1.0
5028    1.0
5375    1.0
Name: result, dtype: float64
        B&#039;T X Neo
        Street Fighter: Aratanaru Kizuna
        Allison to Lillia
        Noragami OVA
        Gene Diver
```