CARTOON RECOMMENDATION SYSTEM

CS6611 - CREATIVE AND INNOVATIVE PROJECT

Submitted by

KOUSHIK RAMESHBABU (2020103019)

PRANAV PRSHANTH (2020103554)

TARINI THUTHIKA (2020103581)

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2023

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **CARTOON RECOMMENDATION SYSTEM** is the bonafide work of **KOUSHIK RAMESHBABU, PRANAV PRSHANTH** and **TARINI THUTHIKA** who carried out the project work under my supervision, for the fulfillment of the requirements as part of the CS6611 – Creative and Innovative Project.

Dr. S. VALLI	**Mrs. C. SUGANTHINI**
PROFESSOR & HEAD	**TEACHING FELLOW**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Department of Computer Science	Department of Computer Science
and Engineering	and Engineering
Anna University	Anna University
Chennai - 600025	Chennai - 600025

ABSTRACT

The Cartoon Recommendation System that combines User Based, Content Based, and KNN models using Stacking is a personalized recommendation system designed to provide relevant and engaging cartoon recommendations to users. The system uses a combination of User Based, Content Based, and KNN models to analyze user preferences and recommend cartoons that match their interests. Stacking is used to combine the predictions of the individual models, leading to more accurate recommendations. The system addresses challenges such as data quality and privacy, overfitting, and the cold-start problem. The system's scope is broad and can be applied in various areas of the animation industry, including streaming platforms, broadcasters, advertisers, production studios, educational institutions, and parental control. The implementation of the system can lead to increased revenue, enhanced user engagement, and improved user satisfaction. Overall, the Cartoon Recommendation System that combines User Based, Content Based, and KNN models using Stacking offers a personalized and engaging viewing experience, meeting the growing demand for personalized content in the highly competitive animation industry.

ACKNOWLEDGEMENT

Foremost, we would like to express our sincere gratitude to our project guide, **Mrs. C. Suganthini**, Teaching Fellow, Department of Computer Science and Engineering, College of Engineering Guindy, Chennai for her constant source of inspiration. We thank her for the continuous support and guidance which was instrumental in taking the project to successful completion.

We are grateful to **Dr. S. Valli**, Professor and Head, Department of Computer Science and Engineering, College of Engineering Guindy, Chennai for her support and for providing necessary facilities to carry out for the project.

We would also like to thank our friends and family for their encouragement and continued support. We would also like to thank the Almighty for giving us the moral strength to accomplish our task.

KOUSHIK	**PRANAV**	**TARINI**
RAMESHBABU	**PRSHANTH**	**THUTHIKA**

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Considering The Cartoon Recommendation System combines various models such as User Based, Content Based, and KNN using Stacking. This system is designed to provide personalized cartoon recommendations based on the viewing history and preferences of individual users. The User Based model analyzes the viewing history of users and recommends cartoons based on their past preferences. The Content Based model recommends cartoons based on the similarity of their attributes, such as genre, animation style, and storyline. The KNN model recommends cartoons based on the similarity of the viewing habits of other users. To further enhance the accuracy of the recommendations, these models are combined using Stacking. Stacking is an ensemble learning technique that combines the predictions of multiple models to improve the accuracy of the final recommendation. Our system takes into account various factors such as the age of the user, their preferences, and the popularity of the cartoons. It also provides the option to filter recommendations based on specific genres, animation styles, or ratings. Our aim is to provide users with an engaging and personalized viewing experience by offering them cartoons that match their preferences and interests.

1.1 OBJECTIVES

The primary objective of this system is to provide personalized recommendations to users based on their viewing history and preferences. The system aims to improve the accuracy of the recommendations by combining multiple models and using ensemble learning techniques like Stacking. The system should provide flexibility to users to filter recommendations based on specific genres, animation styles, or ratings. The system aims to enhance user satisfaction by offering a more engaging and personalized viewing experience. The system should provide diverse recommendations to users, ensuring that they are exposed to a wide range of cartoons and animation styles. The system should be scalable and able to handle large volumes of user data and provide recommendations in real-time. The system should be easy to maintain and update as per the changing user preferences and trends in the animation industry. By achieving these objectives, the Cartoon Recommendation System will provide an excellent user experience and help to build a loyal user base.

1.2 PROBLEM STATEMENT

For building a recommender system from scratch, we face several different problems. Currently there are a lot of recommender systems based on the user information, so what should we do if the website has not gotten enough users. After that, we will solve the representation of a cartoon, which is how a system can understand a cartoon. That is the precondition for comparing similarity between two cartoon. Cartoon features such as genre, actor and director is a way that can categorize cartoon. But for each feature of the cartoon, there should be different weight for them and each of them plays a different role for recommendation.

1.3 NEED FOR THE SYSTEM

In today's world, consumers demand personalized content that matches their preferences and interests. The Cartoon Recommendation System meets this need by providing personalized cartoon recommendations to users. With the rise of online streaming platforms and increased access to high-speed internet, there is a massive amount of data generated every day. The Cartoon Recommendation System can analyze this data and provide accurate recommendations to users.

1.4 CHALLENGES OF THE SYSTEM

The accuracy of the recommendations is highly dependent on the quality and quantity of data available. Therefore, ensuring the quality and quantity of data is a significant challenge. Collecting and storing user data raises privacy concerns. Therefore, it is essential to ensure that the system complies with data privacy laws and regulations. Stacking and other ensemble learning techniques used in the system can lead to overfitting. Therefore, it is essential to tune the hyperparameters of the models and regularize them to prevent overfitting. While the system aims to enhance user satisfaction, it is challenging to predict user preferences accurately. There is a risk that users may not be satisfied with the recommendations provided by the system.The system must be designed to handle large volumes of data and provide recommendations in real-time. Therefore, it is essential to ensure that the system is scalable and can handle increasing Data.

1.5 SCOPE OF THE PROJECT

The system can be implemented by streaming platforms to provide personalized recommendations to users. This can enhance user engagement and encourage users to spend more time on the platform, leading to increased revenue. Broadcasters can use the system to recommend cartoons and animation shows to their viewers, leading to increased viewership and loyalty. Advertisers can use the system to identify cartoons and animation shows that match the interests of their target audience. This can improve the effectiveness of their advertising campaigns. Production studios can use the system to analyze user preferences and trends in the animation industry to produce new content that matches user preferences. Educational institutions can use the system to recommend educational cartoons and animation shows to students that match their learning needs and interests. The system can be used by parents to filter out inappropriate content and recommend suitable cartoons and animation shows for their children.

CHAPTER 2
LITERATURE SURVEY

2.1 ANALYSING USER SOCIAL NETWORK

Paper [1] proposes data collection mechanism and predictive models that leverage language and behavioural patterns on Twitter to detect mental illness in an online environment. It offers an analytics web-service to explore linguistic and behavioural patterns of social media users in relation to mental illnesses. The system developed over this offers real-time analytics. To train the model, the users are distinguished as either - patient and non-patient, this can be further extended to users such as doctors or therapists. Moreover, the final prediction is done only between two mental disorders - borderline personality disorder and bipolar disorder.

2.2 USING LSTM AND CNN

Paper [2] proposes LSTM component will be used to model the temporal dynamics of user preferences for cartoon. LSTM is a type of recurrent neural network that is well-suited for modelling sequences of data over time. It has been successfully used in various natural language processing and time series analysis applications. LSTM and CNN are powerful deep learning techniques that have been shown to improve the accuracy of recommendation models and by providing more accurate recommendations, users are more likely to find cartoon that they enjoy and are more likely to continue using the platform. Disadvantage of using LSTM and CNN for cartoon recommendation is the time required to train these models and LSTM and CNN models are prone to overfitting, which means they can become too.

2.3 CONSTRUCTION USING PYTHON

Paper [3] proposes the construction of a cartoon recommendation system based on Python involves developing a system that suggests cartoon to users based on their preferences and viewing history. This system utilizes machine learning algorithms and data analysis techniques to analyze user behavior and provide personalized cartoon recommendations. Python is a high-level programming language that is easy to learn and use, making it accessible for developers of different skill levels. It has a simple syntax and a large user community that offers extensive documentation, tutorials, and support and also Python is a scalable language that can handle large data sets and complex computations. This makes it suitable for building recommendation systems that can handle millions of users and items. Python is an interpreted language, which can result in slower performance compared to compiled languages such as C++ or Java. As with any software system, a cartoon recommendation system based on Python will require ongoing maintenance and updates. This can be a challenge, especially for businesses with limited resources and expertise.

2.4 HYBRID MODEL

Paper [4] proposes a hybrid approach to cartoon recommendation that combines collaborative filtering and content-based filtering. Collaborative filtering is a technique that uses the preferences of similar users to generate recommendations. Content-based filtering, on the other hand, analyses the attributes of items, such as the genre, actors, and plot, to recommend similar items. The Orbit system also has the potential to provide more diverse recommendations. Because it considers both user behavior and current trends, it can recommend cartoon that the user may not have considered before, leading to a more varied cartoon-watching experience and the improved accuracy and diversity of recommendations provided by the Orbit system can lead to a better overall user experience. Users are more likely to find cartoon that they enjoy, leading to increased satisfaction with the recommendation system. The Orbit system's recommendations may be biased towards popular cartoon or those with a high budget. This could result in a lack of diversity in recommendations, with the same cartoon being recommended to many users. And The Orbit system relies on user data, such as ratings and watch history, to generate recommendations. This could raise privacy concerns among users who are uncomfortable with their data being used for marketing purposes.

2.5 **VOTED CLASSIFIER**

Paper [5] proposes the adaptive voting classifier is then introduced, which combines the predictions of multiple base classifiers using a weighted voting scheme. The weights assigned to each base classifier are adaptively adjusted based on their performance on the training data, which allows the classifier to learn from its mistakes and improve its accuracy over time. The weights assigned to each base classifier in the adaptive voting classifier are adaptively adjusted based on their performance on the training data. This means that the model can learn from its mistakes and improve its accuracy over time, making it adaptable to changes in the cartoon industry. The proposed approach allows for interpretability by identifying the key features that contribute to the prediction of cartoon quality. This can help cartoon producers and distributors understand what factors influence the quality of a cartoon and make informed decisions. The accuracy of the proposed approach relies on the quality and representativeness of the cartoon dataset used for training and testing. If the dataset is biased or incomplete, the predictions of the model may be less accurate or even misleading. The proposed approach relies on a set of predetermined features to predict the quality of a cartoon. While these features may be important, there may be other factors that are not included in the model that could also influence the quality of a cartoon.

2.6 SUMMARY

As every model has its own advantages and disadvantages it is inferred that combining them would give a better result and more personalized recommendation.

CHAPTER 3

SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

Shown below is the system block diagram consisting of 6 modules of the recommendation system.

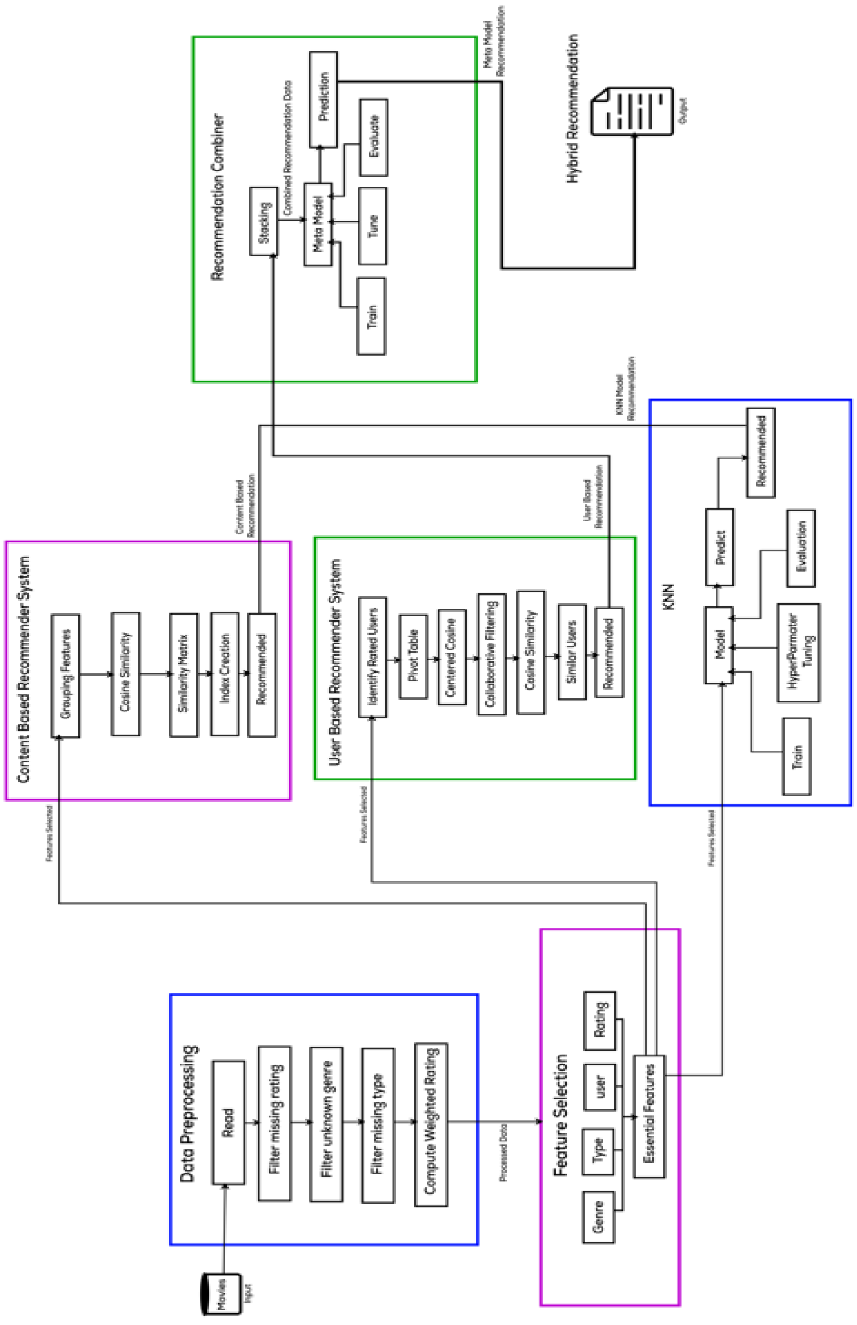


Figure 3.1 Recommendation System Diagram

3.2 SYSTEM REQUIREMENTS

3.2.1. Functional Requirements

3.2.1.1. Hardware Requirements

Since the application must run over the internet, all the hardware required to be connected to the internet will be a hardware interface for the system. As for e.g. Modem, WAN – LAN, Ethernet Cross-Cable. The system should require hardware that is capable of storing large amounts of data. It should have sufficient storage space to store the collected social media data and analysis results. The system should require hardware that is capable of processing large amounts of data quickly and efficiently. It should have a fast processor and sufficient RAM to perform the required data analysis. The system should require output devices such as monitors or screens for users to view the software interface. The output devices should be compatible with the hardware and should have sufficient resolution to display the analysis results clearly. The system should have a backup and recovery system in place to ensure data is not lost in case of hardware failure. The hardware should have sufficient backup and recovery capabilities to ensure the system can be restored quickly and easily. The system should be compatible with mobile devices such as smartphones and tablets. The software interface should be optimized for mobile devices to enable users to access the platform from anywhere

3.2.1.2. Software Requirements

We have chosen Windows operating system for its best support and User friendliness. To save the movie details, users preferences etc, we have chosen SQL database. To implement the project we will use Python libraries,Javascript,Html etc. The system should generate personalized movie recommendations for each user based on their past ratings and other data. This could involve user-based, content-based, or hybrid recommendation algorithms. Users should be able to filter their recommendations based on criteria such as genre, release year, or average rating. Users should be able to provide feedback on their recommended movies, such as marking them as watched or rating them. The system should monitor its performance in terms of recommendation accuracy, user engagement, and server load, and alert administrators of any issues

3.2.2. Non- Functional Requirements

3.2.2.1. Performance

The project shall be based on the web and has to be run from a web server. The product shall take initial load time depending on internet connection strength which also depends on the media from which the product is run. The system should be able to collect user data quickly and efficiently. It should be capable of processing large amounts of data in real-time. The system should be scalable to handle a large volume of users and data. It should be able to handle an increasing number of users and data without affecting performance. The system should be available 24/7. It should be accessible to users at all times, without any significant downtime or maintenance. It must also be reliable. The system should be compatible with different operating systems and web browsers. The software interface should work seamlessly across different devices and platform.

CHAPTER 4

MODULE DESCRIPTION

4.1 DATA PREPROCESSING

Load the raw cartoon dataset and then remove any duplicate entries. Remove any cartoon with missing data and split the cartoon title and year of release into separate columns. Remove any irrelevant columns such as cartoon poster or synopsis. Clean the data by removing special characters, symbols, and other non-alphanumeric characters. cleaned, normalized, and had any noise removed, it can be used to create a user-cartoon rating matrix, where each row represents a user, and each column represents a cartoon. The matrix has ratings for each user-cartoon combination, with missing ratings represented as NaN values. This matrix can be split into training and testing datasets, which are used to evaluate the performance of the recommendation algorithm. Finally, the preprocessed data can be saved to a new file for later use in the recommendation system.

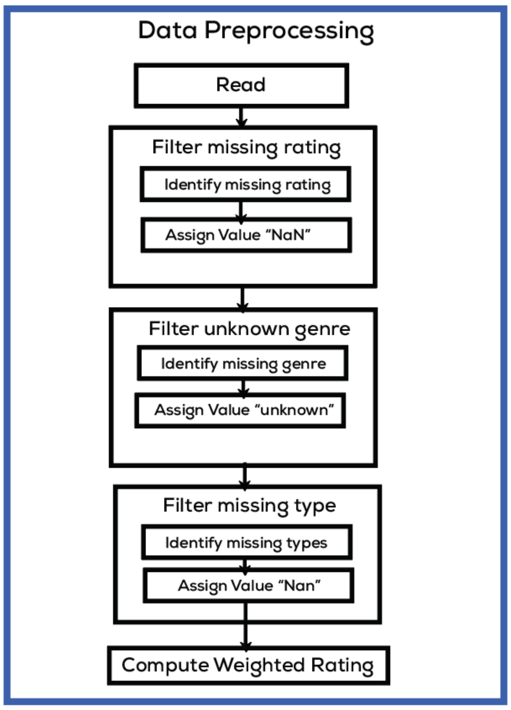


Figure 4.1 Data preprocessing

Input: Cartoon.csv

Output: Processed Data

Pseudocode:

```
df = pd.read_csv("anime.csv")   df.head()
df.loc[df['rating'].isnull(), 'rating'] = 0.0
df['rating'].isnull().any
df['type'].value_counts().plot.bar()
df.loc[(df['episodes']=="Unknown") & (df['type'].isnull())].head(
df.loc[(df['name'] == "Citrus"), 'episodes'] = '12'
df.loc[(df['name'] == "Hitorijime My Hero"), 'type'] = 'TV'
df.loc[(df['name'] == "Hitorijime My Hero"), 'episodes'] = '12'
df.isnull().sum()
```

4.2 FEATURE SELECTION

Information Gain measures the reduction in entropy or uncertainty of the target variable (in this case, the user's rating of a cartoon) when a feature is added to the model. The higher the Information Gain, the more important the feature is. In a cartoon recommendation system, features such as the cartoon genre, director, lead actor, and release year could be evaluated using Information Gain to determine their relevance in predicting a user's rating. The Gain Ratio technique is an extension of Information Gain that considers the intrinsic information contained in a feature. It evaluates the Information Gain of a feature relative to its intrinsic information, which is calculated based on the number of categories or values that the feature can take. Gain Ratio is a more robust measure of feature importance that can help avoid overfitting. In a cartoon recommendation system, features such as user demographics, cartoon popularity, and user preferences could be evaluated using Gain Ratio to determine their relevance in predicting a user's rating.

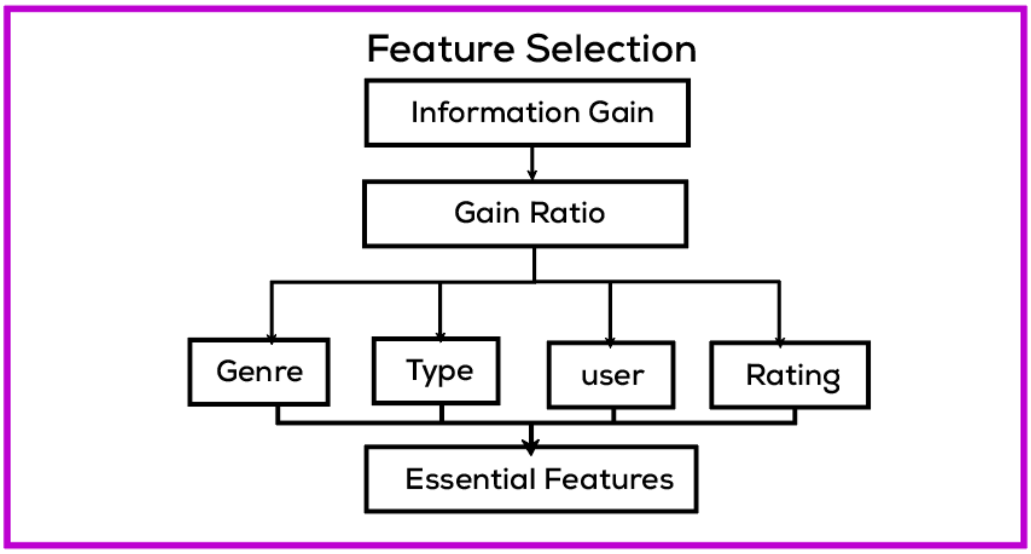


Figure 4.2 Feature Selection

Input: Processed Data

Output: Features Selected

Pseudocode:

```
def calculate_entropy(data):
total_instances = len(data)
positive_instances = len(data[data['community_rating'] >= 7])
negative_instances = len(data[data['community_rating'] < 7])
if positive_instances == 0 or negative_instances == 0:
return 0
positive_probability = positive_instances / total_instances
negative_probability = negative_instances / total_instances
```

4.3 CONTENT BASED RECOMMENDER SYSTEM

Content-based Recommender is a technique that recommends cartoons to users based on the similarity of the cartoon attributes and the user's preferences. This approach analyzes the attributes of cartoons such as genre, director, actors, and other metadata and recommends cartoons that are like the ones the user has enjoyed in the past. We use here, Cosine similarity score: This method measures the similarity between two cartoon based on their features and calculates the cosine of the angle between them.

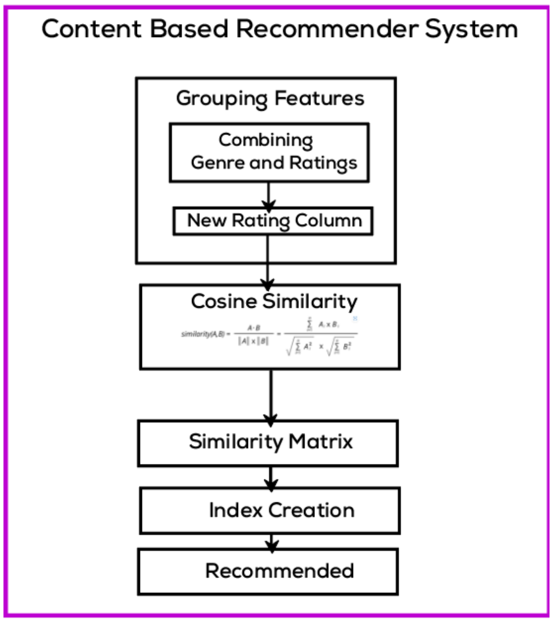


Figure 4.3 Content Based Recommendation System

Input: Feature Selected

Output: Content Based Recommendation

Pseudocode:

```
def get_recommendation(anime_name,
similarity=cosine_sim):

idx = anime_index[anime_name]

sim_scores = list(enumerate(cosine_sim[idx]))

sim_scores = sorted(sim_scores, key=lambda x: x[1],
reverse=True)

sim_scores = sim_scores[0:11]

anime_indices = [i[0] for i in sim_scores]
```

4.4 User Based Recommender System

A user-based cartoon recommendation system is a type of collaborative filtering algorithm that suggests cartoons to a user based on their similarity to other users. The idea behind this approach is that users who have similar cartoon preferences will like similar cartoons. Gather data: Collect data on user ratings for different cartoon. Create a user-item matrix by converting the user ratings data into a user-item matrix, where each row represents a user, each column represents a cartoon, and each cell represents a rating. Calculate the similarity between each pair of users using a similarity metric such as cosine similarity, Pearson correlation coefficient, or Jaccard similarity. This step involves comparing the rating patterns of each pair of users and calculating how closely they match. Present the recommendations to the user, either in a list or in the form of a personalized cartoon suggestion.

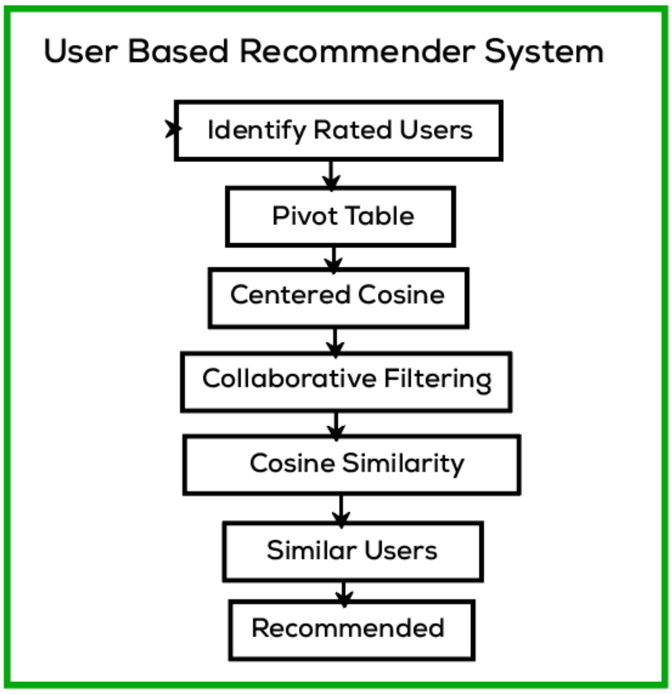


Figure 4.4 User Based Recommendation System

Input: Features Selected

Output: User Based Recommendation

Pseudocode:

```
matrix = load_data()
num_users = number_of_users(matrix)
for i in range(num_users):
    similarities = []
    for j in range(num_users):
        if i != j:
            sim = similarity(matrix[i], matrix[j])
            similarities.append((j, sim))
    similarities.sort(key=lambda x: x[1], reverse=True)
```

4.5 KNN MODEL

In a cartoon recommendation system, the training and testing data is used to evaluate the performance of the recommendation algorithm. The goal of the recommendation system is to predict the cartoon preferences of users based on their historical cartoon ratings and other features. The training data is used to train the recommendation algorithm using a portion of the available historical cartoon ratings. The remaining portion of the historical cartoon ratings is used as the testing data to evaluate the accuracy of the recommendation algorithm. The training and testing data should be representative of the population of users who will be using the recommendation system. This means that the data should cover a range of user preferences and demographics to ensure that the recommendation algorithm can generalize well to new users. It is important to ensure that the training and testing data are not biased towards a particular subset of users or cartoon, as this can lead to inaccurate recommendations. The data is typically split randomly into two sets using a ratio that depends on the size of the dataset and the complexity of the recommendation algorithm.

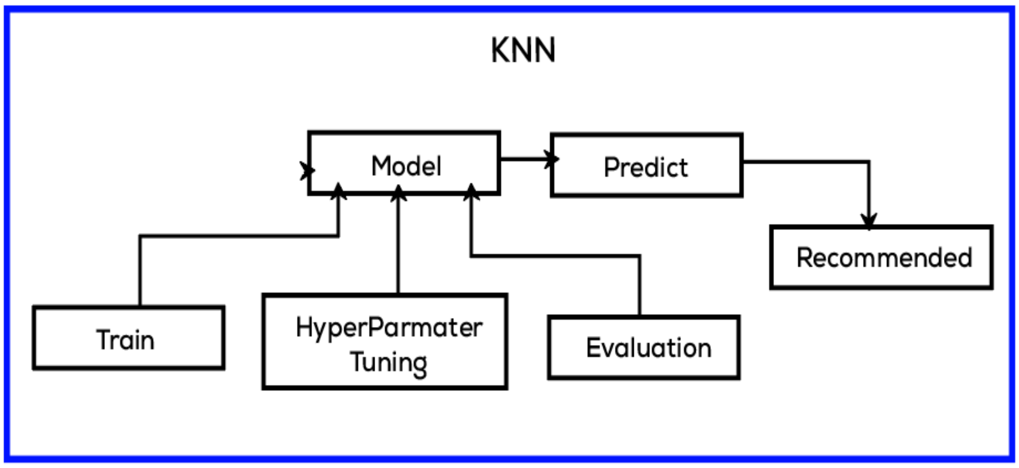


Figure 4.5 KNN Model

Input: Feature Selected

Output: KNN Model Recommendation

Pseudocode:

```
X_train, X_test, y_train, y_test = split_data(matrix)
model = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
model.fit(X_train, y_train)
user = get_user_input()
user_ratings = matrix[user]
neighbors = model.kneighbors([user_ratings], return_distance=False)
recommended_cartoon = {}
```

4.6 Recommendation Combiner

Recommendation stacking is a technique used in cartoon recommendation systems where multiple models are trained and their recommendations are combined to improve the accuracy and diversity of the recommendations. Here's a pseudocode for a recommendation stacking-based cartoon recommendation system. Load the user ratings data into a user-item matrix. Remove duplicates, missing values, and outliers from the data. Split the data into training and testing sets. Train multiple base models on the training set using different algorithms such as KNN, matrix factorization, or neural networks. Generate base model recommendations: For each user in the testing set, generate recommendations using the trained base models.Train a meta-model on the base model recommendations using a machine learning algorithm such as logistic regression or random forest. For a given user, generate recommendations using the trained base models and combine them using the trained meta-model to generate the final recommendations. Evaluate the performance of the system on the testing set using metrics such as precision, recall, and F1-score.

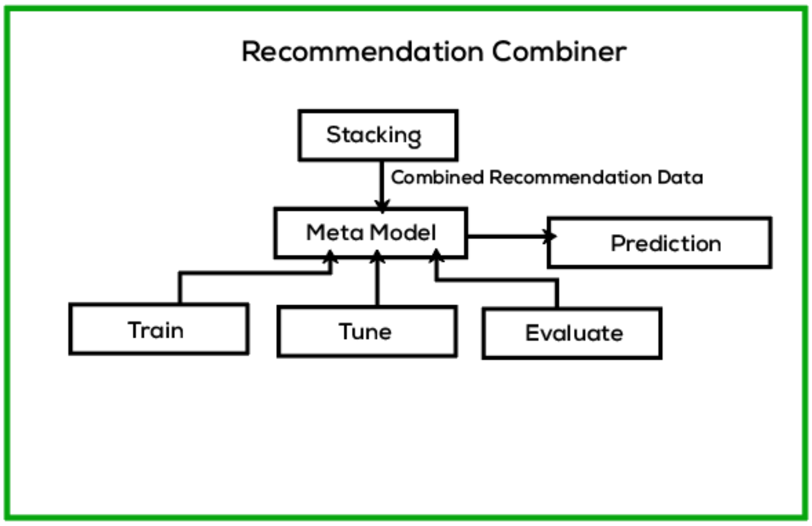


Figure 4.6 Recommendation Combiner

Input: User Based Recommendation

Output: Meta Model Recommendation

Pseudocode:

```
from sklearn.neighbors import KNeighborsClassifier

from sklearn.decomposition import NMF

matrix = load_data()

X_train, X_test, y_train, y_test = split_data(matrix)

knn_model = KNeighborsClassifier(n_neighbors=k, metric='euclidean')

knn_model.fit(X_train, y_train)

nn_model.add(Dense(units=1, activation='sigmoid'))

nn_model.compile(loss='binary_crossentropy', optimizer='adam')

nn_model.fit(X_train, y_train, epochs=num_epochs, batch_size=batch_size)

base_model_recommendations = {}
```

4.7 INNOVATION

I. The use of stacking (Used in Module 6 Recommendation Combiner)

Stacking is a technique used in recommendation systems to combine the predictions of multiple machine learning models to generate more accurate and diverse recommendations. In stacking, several base models are trained on the same data, each using a different set of features or algorithms. The predictions from these base models are then combined using a meta-model, which generates the final recommendation. Stacking is particularly useful when dealing with large and complex datasets where a single model may not be able to capture all of the patterns and nuances in the data. By combining the predictions of multiple models, stacking can improve the accuracy and robustness of the recommendation system. Despite these challenges, stacking has proven to be a powerful technique for improving the performance of recommendation systems. It has been successfully applied in a variety of domains, including e-commerce, social media, and online advertising.

CHAPTER 5

RESULTS AND IMPLEMENTATION

5.1 DATA PREPROCESSING

DATASET DESCRIPTION (anime.csv)

The anime.csv dataset contains information about anime shows. Here is a description of the columns in the dataset: anime_id which is a unique identifier for each anime show. Name which is the name of the anime show. Genre which is the genre(s) of the anime, separated by commas. Type which is the type of the anime, such as TV series, Movie, OVA (Original Video Animation), etc. Episodes which is the number of episodes in the anime. Rating which is the average rating given to the anime on a scale of 0-10. Members which is the number of community members who have added the anime to their list. These columns provide information about the characteristics of each anime show, including its genre, type, number of episodes, average rating, and popularity among community members genre.

Code:

```
df = pd.read_csv("anime.csv")
df.head()
```

Output:

Table 5.1 Anime.csv

anime_id	name	genre	type	episodes	rating	members
32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630
5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26	793665
28977	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.25	114262
9253	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572
9969	Gintama&#039;	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.16	151266

5.1.1 Handling Missing Values

5.1.1.1 Handling Missing Ratings

The movies which have null ratings will be changed to 0. Then we search again to check if there are any other movies with no ratings.

Code:

```
df.loc[df['rating'].isnull(), 'rating'] = 0.0
df['rating'].isnull().any()
```

Output:

```
False
```

5.1.2.2 Handling Missing Type

Code:

```
df['type'].value_counts().plot.bar()
```

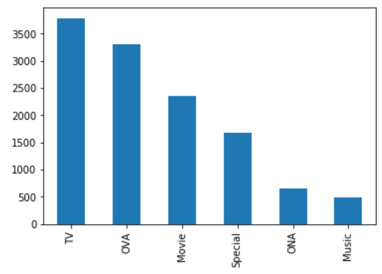


Fig 5.1 genre classification

The graph above represents the type of movies we have. Our approach here is to mark the missing types as “unknown”. Then we check if we can find any null types.

Code:

```
df.loc[(df['episodes']=="Unknown") & (df['type'].isnull())].head()
```

Output:

Table 5.2 After Handling Missing Type

	anime_id	name	genre	type	episodes	rating
10898	30484	Steins;Gate 0	Sci-Fi, Thriller	#N/A	Unknown	0
10900	34437	Code Geass: Fukkatsu no Lelouch	Action, Drama, Mecha, Military, Sci-Fi, Super ...	#N/A	Unknown	0
10906	33352	Violet Evergarden	Drama, Fantasy	#N/A	Unknown	0
10907	33248	K: Seven Stories	Action, Drama, Super Power, Supernatural	#N/A	Unknown	0
10918	33845	Free! (Shinsaku)	School, Sports	#N/A	Unknown	0

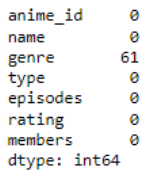
5.1.2.3 Handling Unknown Genre

Fill “Unknown” in movies where there is no genre. Check for null if there are any movies left with null in their genre column

Code:

df.isnull().sum()

Output:



```
anime_id     0
name         0
genre       61
type         0
episodes     0
rating       0
members      0
dtype: int64
```

Code:

df[df['genre'].isnull()]

Output:

Table 5.3 After Handling Unknown Genre

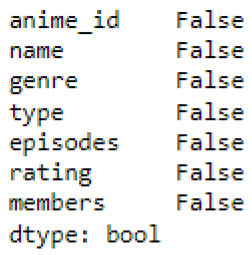
	anime_id	name	genre	type	episodes	rating
2844	33242	IS: Infinite Stratos 2 - Infinite Wedding	#N/A	Special	1	7.15
3541	33589	ViVid Strike!	#N/A	TV	12	6.96
6040	29765	Metropolis (2009)	#N/A	Movie	1	6.27

6646	32695	Match Shoujo	#N/A	ONA	1	6.02
7018	33187	Katsudou Shashin	#N/A	Movie	1	5.79
...	...	...	...	...	...	...
11070	32032	Seikaisuru Kado	#N/A	TV	12	0
11086	34310	Tamagotchi Movie: Tanpen Himitsu no Otodoke Da...	#N/A	Movie	1	0
11097	34474	Tsukipro The Animation	#N/A	TV	Unknown	0
11112	33035	Yuyushiki Special	#N/A	Special	1	0
11113	33390	Zunda Horizon	#N/A	Movie	1	0

Code:

```
df['genre'].fillna('Unknown', inplace=True)
df.isnull().any()
```

Output:



```
anime_id    False
name        False
genre       False
type        False
episodes    False
rating      False
members     False
dtype: bool
```

5.1.3 Calculate Weighted Rating

Weighted ratings are a common approach to building recommendation systems, including those for anime. The basic idea behind a weighted rating system is to take into account both the average rating of a piece of content and the number of ratings it has received. It is done to show the weighted rating to the user whenever they query for similar movies. We can improve the recommendation by sorting the recommendation based on their respective weighted rating.

(weighted rating) = ((v / (v + m)) * R) + ((m / (v + m)) * C)

where:

R = the average rating for the anime

v = the number of ratings for the anime

m = a minimum number of ratings required to be included in the recommendation system (this is used to avoid recommending anime with only a few ratings)

C = the average rating across all anime in the dataset

This approach helps to ensure that anime with a high average rating and a large number of ratings are given priority in the recommendation system, while still taking into account anime with a lower number of ratings.

Code:

```
m = df.members.quantile(0.75)
C = df.rating.mean()
print(m, C)
```

Output:

```
9448.5 6.355887858072929
```

Code:

```
def weighted_rating(df, m, C):
term = df['members'] / (m + df['members'])
return df['rating'] * term + (1-term) * C
df['community_rating'] = df.apply(weighted_rating, axis=1, args=(m,C))
df.head()
```

Output:

Table 5.4 Added Community Rating

	name	genre	type	community_rating
0	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	9.234437
1	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	9.225834
2	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	9.02896
3	Steins;Gate	Sci-Fi, Thriller	TV	9.131071
4	Gintama&#039;	Action, Comedy, Historical, Parody, Samurai, S...	TV	8.995145

5.2 FEATURE SELECTION

Combining essential features which can be used for the models Features like [Genre,type,user,rating] are used in order to train our 3 models proposed in our architectural diagram

Code:

```
import math
def calculate_entropy(data):
total_instances = len(data)
          positive_instances = len(data[data['community_rating'] >= 7])
 negative_instances = len(data[data['community_rating'] < 7])
 if positive_instances == 0 or negative_instances == 0:
  return 0
  positive_probability = positive_instances / total_instances
  negative_probability = negative_instances / total_instances
 entropy = -positive_probability * math.log2(positive_probability) - negative_probability * math.log2(negative_probability)
  return entropy
def calculate_information_gain(data, column):
   entropy = calculate_entropy(data)
   unique_values = data[column].unique()
   weighted_entropy = 0
         for value in unique_values:
      subset = data[data[column] == value]
      subset_entropy = calculate_entropy(subset)
      subset_probability = len(subset) / len(data)
      weighted_entropy += subset_probability * subset_entropy
   information_gain = entropy - weighted_entropy
   return information_gain
```

Output:

```
information_gain = calculate_information_gain(data, 'name')
information_gain
        0.6132511429890424
information_gain = calculate_information_gain(data, 'genre')
information_gain
0.3888641338573728
information_gain = calculate_information_gain(data, 'episodes')
information_gain
0.07391778852040642
information_gain = calculate_information_gain(data, 'type')
information_gain
0.047954103539015414
```

5.2.1 Dropping unused columns

These are the features that will be dropped:

- anime_id -> just the index of the anime, it is easier if we used the panda's index
- rating -> we have weighted rating
- members -> the data is not necessary
- episodes -> the data is not necessary

Code:

```
df.drop(['anime_id', 'rating', 'members', 'episodes'], axis=1, inplace=True)

df.head()
```

Output:

Table 5.5 After Dropping Columns

	name	genre	type	community_rating
0	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	9.234437
1	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	9.225834
2	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	9.02896
3	Steins;Gate	Sci-Fi, Thriller	TV	9.131071
4	Gintama&#039;	Action, Comedy, Historical, Parody, Samurai, S...	TV	8.995145

5.3 CONTENT BASED RECOMMENDATION

5.3.1 Break Down Features

We want our algorithm to treat the 'type' and 'genre' of anime as equal. Therefore if we use label encoding, maybe the algorithm will treat a certain category more important than the other categories.

Code:

```
df = pd.concat([df,
df['type'].str.get_dummies(), df['genre'].str.get_dummies(sep=',')], axis=1)
df.head()
```

Table 5.6 After Breaking Down Features

name	genre	type	community_rating
Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	9.234437
Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	9.225834
Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	9.02896
Steins;Gate	Sci-Fi, Thriller	TV	9.131071
Gintama&#039;	Action, Comedy, Historical, Parody, Samurai, S...	TV	8.995145

5.3.2 Calculate the Similarity Matrix

Represent the ratings data as a matrix, where each row corresponds to a user and each column corresponds to an anime. We calculate the similarity between pairs of anime using a similarity metric called cosine similarity.

cosine_similarity(A, B) = (A . B) / (||A|| * ||B||)

Where A and B are vectors representing the ratings for two different anime, and ||A|| and ||B|| are the magnitudes of those vectors.

Fill in the similarity matrix with the calculated similarity values for each pair of anime.

We use the similarity matrix to make recommendations based on which anime are most similar.

Code:

```
cosine_sim = cosine_similarity(anime_features.values,
anime_features.values)

cosine_sim
```

Output:

```
array([[1.        , 0.        , 0.        , ..., 0.        , 0.        , 0.31622777],
[0.        , 1.        , 0.375     , ..., 0.        , 0.        ,  0.        ],
 [0.        , 0.375     , 1.        , ..., 0.        , 0.        ,  0.        ],      ...,
 [0.        , 0.        , 0.        , ..., 1.        , 1.        , 0.5       ],
 [0.        , 0.        , 0.        , ..., 1.        , 1.        ,0.5       ],
 [0.31622777, 0.        , 0.        , ..., 0.5       , 0.5       ,1.        ]])
```

Code:

```
cosine_sim.shape
```

Output:

```
(12288, 12288)
```

5.3.3 Index Creation

Index Creation is important as it helps to retrieve information easily. It is her done so to view the similar movies given a particular movie name it retrieves the similar movies related to this. We create get_recommendation() to retrieve similar movies given its title.

Code:

```
anime_index = pd.Series(df.index, index=df.name).drop_duplicates()

def get_recommendation(anime_name, similarity=cosine_sim):

idx = anime_index[anime_name]

sim_scores = list(enumerate(cosine_sim[idx]))

sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

sim_scores = sim_scores[0:11]

anime_indices = [i[0] for i in sim_scores]
```

```
 result = df[['name', 'genre',
'community_rating']].iloc[anime_indices].drop(idx)
return result
```

5.3.4 Get Recommendation for a particular title

Here we have searched movie titles and similar movies and tv shows are printed below.

Code:

```
get_recommendation("Steins;Gate")
```

Table 5.7 Content Based Recommendation

	name	genre	community_rating
10898	Steins;Gate 0	Sci-Fi, Thriller	0.852459
3581	Fireball Charming	Sci-Fi	6.574226
7984	Hanoka	Sci-Fi	6.04004
8910	Hoshi no Ko Poron	Sci-Fi	6.360831
10079	RoboDz	Sci-Fi	6.338184
10858	Yuusei Kamen	Sci-Fi	6.356795
10975	Escha Chron	Sci-Fi	5.861462
59	Steins;Gate Movie: Fuka Ryouiki no Déjà vu	Sci-Fi, Thriller	8.504498
126	Steins;Gate: Oukoubakko no Poriomania	Sci-Fi, Thriller	8.34236
196	Steins;Gate: Kyoukaimenjou no Missing Link - D...	Sci-Fi, Thriller	7.946121

5.4 User Based Recommender System

5.4.1 Reading anime.csv and rating.csv

The anime.csv and rating.csv containing the movies,tv cartoon list and their respective ratings is read and printed.

Code:

```
anime = pd.read_csv("anime.csv")
anime.head()
```

Output:

Table 5.8 User Based Recommendation

	anime_id	name	genre	type	episodes	rating
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37
1	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26
2	28977	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.25
3	9253	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17
4	9969	Gintama&#039;	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.16

Code:

```
print(anime.shape)
anime = anime[(anime['type'] == 'TV') | (anime['type'] =='Movie')]
print(anime.shape)
```

Output:

```
(12294, 7)
(6135, 7)
```

Code:

```
m = anime['members'].quantile(0.75)
anime = anime[(anime['members'] >= m)]
anime.shape
```

Output:

```
(1534, 7)
```

Code:

```
rating = pd.read_csv("rating.csv")
rating.head()
```

Output:

Table 5.9 Rating Table

	user_id	anime_id	rating
0	1	20	-1
1	1	24	-1
2	1	79	-1
3	1	226	-1
4	1	241	-1

5.4.2 Creating Index for Anime name

Index Creation is important as it helps to retrieve information easily. Here it is done so as to join the two tables in the next step.

Code:

```
anime_index = pd.Series(anime.index, index=anime.name)
anime_index.head()
```

Output:

```
name
Kimi no Na wa.                        0
Fullmetal Alchemist: Brotherhood      1
Gintama°                              2
Steins;Gate                           3
Gintama&#039;                         4
dtype: int64
```

5.4.3 Joined Data

We (inner)join anime.csv with ratings.csv so now we have the combined table which gives information about an anime and the rating a particular user gave.

Code:

```
joined = anime.merge(rating, how='inner', on='anime_id')
joined.head()
```

Output:

Table 5.10 Joined Table

anime_id	name	genre	type	rating_x	user_id	rating_y
32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	9.37	99	5
32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	9.37	152	10
32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	9.37	244	10
32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	9.37	271	10
32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	9.37	278	#N/A

5.4.4 Pivot Table

A pivot table is a statistics tool that summarizes and reorganizes selected columns and rows of data in a spreadsheet or database table to obtain desired data which can be useful for the model. Here we take in the columns “user-id”,“name”,”rating_y”.

Code:

```
joined = joined[['user_id', 'name', 'rating_y']]
joined = joined[(joined['user_id'] <= 10000)]
pivot = pd.pivot_table(joined, index='user_id', columns='name',
values='rating_y')
pivot.head()
```

Output:

Table 5.11 Pivot Table

name	&quot;Bungaku Shoujo&quot; Movie	.hack//Sign	.hack//Tasogare no Udewa Densetsu	11eyes	30-sai no Hoken Taiiku
user_id					
1	#N/A	#N/A	#N/A	#N/A	#N/A
2	#N/A	#N/A	#N/A	#N/A	#N/A
3	#N/A	#N/A	#N/A	#N/A	#N/A
5	#N/A	#N/A	#N/A	#N/A	#N/A
7	#N/A	#N/A	#N/A	#N/A	#N/A

5.4.5 Drop all users that never rate anime

There is no need for users who have never rated any anime hence we drop those rows as they add no value to our model.

Code:

```
pivot.dropna(axis=0, how='all', inplace=True)
pivot.head()
```

Table 5.12 After Dropping Users

name	&quot;Bungaku Shoujo&quot; Movie	.hack//Roots	.hack//Sign	07-Ghost	11eyes	30-sai no Hoken Taiiku
user_id						
1	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
2	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
3	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
5	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
7	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A

5.4.6 Center the mean around 0

The idea behind centering the mean is to adjust the user ratings so that they reflect the user's preferences relative to the average rating for each anime title.

For each user rating, subtract the mean rating for that anime title. This will adjust the rating so that it reflects the user's preference relative to the average rating.

Divide the centered ratings by the standard deviation for each anime title. This will scale the ratings so that they have a standard deviation of 1, which makes it easier to compare the ratings across different anime titles.

This powerful technique for improving the accuracy of collaborative filtering recommendation systems.

Code:

```
pivot_norm = pivot.apply(lambda x: x - np.nanmean(x), axis=1)
pivot_norm.head()
```

5.4.7 User Based Collaborative Filtering

The idea behind user-based collaborative filtering is to recommend anime titles to a user based on the preferences of other users who are similar to them.

Code:

```
pivot_norm.fillna(0, inplace=True)

pivot_norm.head()
```

5.4.8 Calculate Similar User

Calculate the cosine similarity between each pair of users based on their centered and normalized ratings. The cosine similarity between two vectors x and y is given by the formula:

cosine_similarity(x, y) = dot(x, y) / (norm(x) * norm(y))

where dot(x, y) is the dot product of x and y, and norm(x) and norm(y) are the Euclidean norms of x and y, respectively.

Code:

```
user_sim_df = pd.DataFrame(cosine_similarity(pivot_norm,
pivot_norm), index=pivot_norm.index, columns=pivot_norm.index)

user_sim_df.head()

ef get_similar_user(user_id):

 if user_id not in pivot_norm.index:

return None, None

else:

  sim_users = user_sim_df.sort_values(by=user_id,
ascending=False).index[1:]

   sim_score = user_sim_df.sort_values(by=user_id,
ascending=False).loc[:, user_id].tolist()[1:]

  return sim_users, sim_score
```

```
users, score = get_similar_user(3)
for x,y in zip(users[:10], score[:10]):
 print("User {} with similarity of {}".format(x, y))
```

Output:

```
User 2986 with similarity of 0.3502463920298145
User 3681 with similarity of 0.3407187927372403
User 3028 with similarity of 0.33997937684120616
User 2411 with similarity of 0.33778981442850053
User 4481 with similarity of 0.3376927737260321
User 1966 with similarity of 0.3360271004231097
User 2038 with similarity of 0.3202469495714798
User 1606 with similarity of 0.3194861993583852
User 656 with similarity of 0.31929123836055745
User 3990 with similarity of 0.31674233937417606
```

5.4.8 Get Recommendation

It checks if the user has information in the first place. Then it calculates 10 nearest users. After filtering the 10 users we check what anime the user recommended and recommend him with a list which he hasn't rated.

Code:

```
def get_recommendation(user_id, n_anime=10):

users, scores = get_similar_user(user_id

if users is None or score is None:

return None

user_arr = np.array([x for x in users[:10]])

sim_arr = np.array([x for x in scores[:10]])
```

```
predicted_rating = np.array([])
for anime_name in pivot_norm.columns:
filtering = pivot_norm[anime_name].loc[user_arr] != 0.0
temp = np.dot(pivot[anime_name].loc[user_arr[filtering]],
sim_arr[filtering]) / np.sum(sim_arr[filtering])
predicted_rating = np.append(predicted_rating, temp)
 temp = pd.DataFrame({'predicted':predicted_rating,
'name':pivot_norm.columns})
filtering = (pivot_norm.loc[user_id] == 0.0)
temp = temp.loc[filtering.values].sort_values(by='predicted',
ascending=False)
return anime.loc[anime_index.loc[temp.name[:n_anime]]]
get_recommendation(3)
```

5.4.9 Get Recommendation for an user id

Code:

```
get_recommendation(3)
```

Output:

Table 5.13 Recommendation for User ID(3)

	anime_id	name	genre	type	episodes	rating
1304	3712	Zero no Tsukaima: Princesses no Rondo	Action, Adventure, Comedy, Ecchi, Fantasy, Har...	TV	12	7.6
3788	10578	C³	Action, Comedy, Ecchi, School, Supernatural	TV	12	6.88
346	223	Dragon Ball	Adventure, Comedy, Fantasy, Martial Arts, Shou...	TV	153	8.16
100	30230	Diamond no Ace: Second Season	Comedy, School, Shounen, Sports	TV	51	8.5

5.5 KNN MODEL

The basic idea behind using k-NN for recommendation is to find the k nearest neighbors (items or users) to a given target item or user and recommend items that are popular among those neighbors.

5.5.1 Training the model

Training the model with n as 6 and using ball_tree algorithm

Code:

```
from sklearn.neighbors import NearestNeighbors

nbrs = NearestNeighbors(n_neighbors=6,
algorithm='ball_tree').fit(anime_features)

distances, indices = nbrs.kneighbors(anime_features)

Distances

Indices
```

Output:

```
array([[0.00000000e+00, 1.01633857e+00, 1.03484164e+00,
1.03547556e+00,
        1.41673596e+00, 1.43504703e+00],
       [0.00000000e+00, 1.02326345e+00, 1.49460533e+00,
1.51670381e+00,
        1.56629658e+00, 1.58703108e+00],
       [0.00000000e+00, 3.78413369e-02, 4.18152040e-02, 2.35264187e-
01,
        3.15115844e-01, 1.41517800e+00],
       ...,
       [0.00000000e+00, 1.66526972e-03, 1.68342172e-03, 5.66357548e-
03,
        6.86823383e-03, 6.86911414e-03],
       [0.00000000e+00, 1.11268583e-03, 1.20660436e-03, 2.22255509e-
03,
        2.24564527e-03, 3.33391240e-03],
       [0.00000000e+00, 1.00000248e+00, 1.00002222e+00,
1.00002225e+00,
        1.00003025e+00, 1.00005000e+00]])
```

```
array([[    0,   208,  1494,  1959,    60,   894],
       [    1,   200,   268,   101,   795,   290],
       [    2,     4,     9,    12, 10896,     8],
       ...,
       [12291, 12238, 12237, 12236, 12256, 12235],
       [12292, 12231, 12232, 12230, 12229, 12283],
       [12293,  7426,  8279,  7349,  7335,  7498]], dtype=int64)
```

5.5.2 Prediction Function

Creating Prediction Function

Code:

```
def get_index_from_name(name):
 return anime[anime["name"]==name].index.tolist()[0]
all_anime_names = list(anime.name.values)
def get_id_from_partial_name(partial):
 for name in all_anime_names:
 if partial in name:
  print(name,all_anime_names.index(name))
def print_similar_animes(query=None,id=None):
if id:
for id in indices[id][1:]:
print(anime.ix[id]["name"])
if query:
found_id = get_index_from_name(query)
for id in indices[found_id][1:]:
print(anime.loc[id]["name"])
```

5.5.3 Testing Prediction

Code:

```
print_similar_animes(query="Naruto")
```

Output:

```
Naruto Shippuuden
Katekyo Hitman Reborn
Bleach
Dragon Ball Z
Boku no Hero Academia
```

5.6. Recommendation Combiner

5.6.1 Getting user recommendation list

Getting user based recommendation by calling the function we have defined before in the user based recommendation system

Code:

```
user_recommendations=get_recommendation(5)
user_recommendations
```

Output:

Table 5.14 Recommendation for user 5

	anime_id	name	genre	type	episodes	rating
0	32281	Kimi no Na wa	Drama, Romance, School, Supernatural	Movie	1	9.37
230	3701	Kaiba	Adventure, Mystery, Romance, Sci-Fi	TV	12	8.29
38	19	Monster	Drama, Horror, Mystery, Police, Psychological,...	TV	74	8.72
91	13125	Shinsekai yori	Drama, Horror, Mystery, Sci-Fi, Supernatural	TV	25	8.53
6	11061	Hunter x Hunter 2011	Action, Adventure, Shounen, Super Power	TV	148	9.13

1	5114	Fullmetal Alchemist Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26
115	523	Tonari no Totoro	Adventure, Comedy, Supernatural	Movie	1	8.48
5	32935	Haikyuu Karasuno Koukou VS Shiratorizawa Gakue...	Comedy, Drama, School, Shounen, Sports	TV	10	9.15
253	1364	Detective Conan Movie 05 Countdown to Heaven	Adventure, Comedy, Mystery, Police, Shounen	Movie	1	8.25
6338	5178	First Squad The Moment of Truth	Action, Historical, Military, Super Power, Sup...	Movie	1	6.16

5.6.2 Getting Search Key

Search Key is the first element in the user based recommendation list

Code:

```
search_key=user_recommendations[0:1]

search_key

current_index = search_key.index.astype(int)

index_value = int(search_key.index[0])

index_value

element = str(search_key.at[index_value,'name'])
element
```

Output:

```
'Kaibutsu Oujo'
```

5.6.2 Getting Content Based recommendation list

Getting Content Based recommendation by calling the function we have defined before in the user based recommendation system

Code:

```
content_recommendations=get_recommendation1(element)
```

Output:

Table 5.15 Content Based Recommendation for user 5

	name	genre	community_rating
981	Mousou Dairinin	Drama, Mystery, Police, Psychological, Superna...	7.651117
6998	Mayoiga	Drama, Horror, Mystery, Psychological	5.841508
199	Death Parade	Drama, Game, Mystery, Psychological, Thriller	8.282582
669	Aoi Bungaku Series	Drama, Historical, Psychological, Seinen, Thri...	7.757694
3806	Night Head Genesis	Drama, Horror, Mystery, Psychological, Superna...	6.716589
6009	Narutaru: Mukuro Naru Hoshi Tama Taru Ko	Drama, Seinen, Thriller	6.306093
53	Rainbow: Nisha Rokubou no Shichinin	Drama, Historical, Seinen, Thriller	8.495083
54	Re:Zero kara Hajimeru Isekai Seikatsu	Drama, Fantasy, Psychological, Thriller	8.580919

5.6.3 Getting KNN recommendation list

Getting KNN based recommendation by calling the function we have defined before in the User Based recommendation system

Code:

```
knn_recommendations=print_similar_animes(query=element)
```

Output:

```
Mousou Dairinin
```

Mayoiga
Death Parade
Aoi Bungaku Series
Night Head Genesis

5.6.4 Getting Combined recommendation list

Code:

```
combined_recommendations = pd.concat([user_recommendations,
content_recommendations, knn_recommendations])
```

Output:

Table 5.16 Combined Table

	anime_id	name	genre	type	episodes	rating
0	32281	Kimi no Na wa	Drama, Romance, School, Supernatural	Movie	1	9.37
230	3701	Kaiba	Adventure, Mystery, Romance, Sci-Fi	TV	12	8.29
38	19	Monster	Drama, Horror, Mystery, Police, Psychological,...	TV	74	8.72
91	13125	Shinsekai yori	Drama, Horror, Mystery, Sci-Fi, Supernatural	TV	25	8.53
6	11061	Hunter x Hunter 2011	Action, Adventure, Shounen, Super Power	TV	148	9.13
1	5114	Fullmetal Alchemist Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26
115	523	Tonari no Totoro	Adventure, Comedy, Supernatural	Movie	1	8.48
5	32935	Haikyuu Karasuno Koukou VS Shiratorizawa Gakue...	Comedy, Drama, School, Shounen, Sports	TV	10	9.15
253	1364	Detective Conan Movie 05 Countdown to Heaven	Adventure, Comedy, Mystery, Police, Shounen	Movie	1	8.25
6338	5178	First Squad The Moment of Truth	Action, Historical, Military, Super Power, Sup...	Movie	1	6.16
981	#N/A	Mousou Dairinin	Drama, Mystery, Police, Psychological, Superna...	#N/A	#N/A	#N/A
6998	#N/A	Mayoiga	Drama, Horror, Mystery, Psychological	#N/A	#N/A	#N/A
199	#N/A	Death Parade	Drama, Game, Mystery, Psychological, Thriller	#N/A	#N/A	#N/A

5.6.5 Assigning Weights

Assign weights to each recommendation

Code:

```
user_weight = 0.4
content_weight = 0.3
knn_weight = 0.3
```

5.6.6 Stacking

Combining the three recommendation list generated by the corresponding models and creating a weighted score column(recommendation score).

Code:

```
combined_recommendations['recommendation_score'] =
(combined_recommendations['rating'] * user_weight) + \
(combined_recommendations['rating'] * content_weight) + \
(combined_recommendations['rating'] * knn_weight)

sorted_recommendations =
combined_recommendations.sort_values('recommendation_score',
ascending=False)
```

5.6.7 Final Prediction

This is the final prediction after combining the three recommendation list.

Code:

```
_recommendations = sorted_recommendations.head(20)
recommended_titles = list(top_recommendations['name'])
recommended_titles
```

Output:

```
['Kimi no Na wa ',
 'Fullmetal Alchemist Brotherhood',
 'Haikyuu Karasuno Koukou VS Shiratorizawa Gakuen Koukou',
 'Hunter x Hunter 2011 ',
 'Monster',
 'Shinsekai yori',
 'Tonari no Totoro',
 'Kaiba',
 'Detective Conan Movie 05 Countdown to Heaven',
 'First Squad The Moment of Truth',
 'Mousou Dairinin',
 'Mayoiga',
 'Death Parade',
 'Aoi Bungaku Series',
 'Night Head Genesis',
 'Narutaru: Mukuro Naru Hoshi Tama Taru Ko',
 'Rainbow: Nisha Rokubou no Shichinin',
'Re:Zero kara Hajimeru Isekai Seikatsu',
 'Mahou Shoujo Madoka★Magica',
 'Gyakkyou Burai Kaiji: Ultimate Survivor']
```

CHAPTER 6

TEST CASES AND PERFORMANCE METRICS

6.1 TEST CASES

6.1.1 Content Based Recommendation

Code:

```
anime_index = pd.Series(df.index, index=df.name).drop_duplicates()
def get_recommendation(anime_name, similarity=cosine_sim):
idx = anime_index[anime_name]
sim_scores = list(enumerate(cosine_sim[idx]))
sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
sim_scores = sim_scores[0:11]
anime_indices = [i[0] for i in sim_scores]
 result = df[['name', 'genre',
'community_rating']].iloc[anime_indices].drop(idx)
return result
```

Output

```
get_recommendation("Kimi no Na wa.")
```

Table 6.1 Test Case 1

	name	genre	community_rating
208	Kokoro ga Sakebitagatterunda.	Drama, Romance, School	8.051436
1494	Harmonie	Drama, School, Supernatural	7.234142
1959	Air Movie	Drama, Romance, Supernatural	7.207802
60	Hotarubi no Mori e	Drama, Romance, Shoujo, Supernatural	8.507055
1199	&quot;Bungaku Shoujo&quot; Movie	Drama, Mystery, Romance, School	7.391296

2103	Clannad Movie	Drama, Fantasy, Romance, School	7.263791
5796	Taifuu no Noruda	Drama, School, Sci-Fi, Supernatural	6.352344
5805	Wind: A Breath of Heart OVA	Drama, Romance, School, Supernatural	6.354841
6394	Wind: A Breath of Heart (TV)	Drama, Romance, School, Supernatural	6.258412
894	Momo e no Tegami	Drama, Supernatural	7.443333

get_recommendation("Kokoro ga Sakebitagatterunda.")

Table 6.2 Test Case 2

	name	**genre**	**community_rating**
0	Kimi no Na wa.	Drama, Romance, School, Supernatural	9.234437
1199	&quot;Bungaku Shoujo&quot; Movie	Drama, Mystery, Romance, School	7.391296
2103	Clannad Movie	Drama, Fantasy, Romance, School	7.263791
5697	Shiranpuri (Movie)	Drama, School	6.36185
10123	Samurai	Drama, Romance	6.323656
1389	Orange: Mirai	Drama, Romance, School, Sci-Fi, Shoujo	7.27137
3544	Ace wo Nerae! (1979)	Drama, Romance, School, Shoujo, Sports	6.440264
11	Koe no Katachi	Drama, School, Shounen	8.823088
265	Kaze Tachinu	Drama, Historical, Romance	8.030089
411	Byousoku 5 Centimeter	Drama, Romance, Slice of Life	8.050585

These are the recommendations based on “content” we have searched and similar content is being displayed.

6.1.2 User Based Recommendation

Code:

```
def get_recommendation(user_id, n_anime=10):
users, scores = get_similar_user(user_id
if users is None or score is None:
return None
user_arr = np.array([x for x in users[:10]])
sim_arr = np.array([x for x in scores[:10]])
predicted_rating = np.array([])
for anime_name in pivot_norm.columns:
filtering = pivot_norm[anime_name].loc[user_arr] != 0.0
temp = np.dot(pivot[anime_name].loc[user_arr[filtering]],
sim_arr[filtering]) / np.sum(sim_arr[filtering])
predicted_rating = np.append(predicted_rating, temp)
 temp = pd.DataFrame({'predicted':predicted_rating,
'name':pivot_norm.columns})
filtering = (pivot_norm.loc[user_id] == 0.0)
temp = temp.loc[filtering.values].sort_values(by='predicted',
ascending=False)
return anime.loc[anime_index.loc[temp.name[:n_anime]]]
get_recommendation(5)
```

Output:

Table 6.3 Test Case 3

	anime_id	name	genre	type	episodes
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1

230	3701	Kaiba	Adventure, Mystery, Romance, Sci-Fi	TV	12
38	19	Monster	Drama, Horror, Mystery, Police, Psychological,...	TV	74
91	13125	Shinsekai yori	Drama, Horror, Mystery, Sci-Fi, Supernatural	TV	25
6	11061	Hunter x Hunter (2011)	Action, Adventure, Shounen, Super Power	TV	148
1	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64
115	523	Tonari no Totoro	Adventure, Comedy, Supernatural	Movie	1
5	32935	Haikyuu!!: Karasuno Koukou VS Shiratorizawa Ga...	Comedy, Drama, School, Shounen, Sports	TV	10
253	1364	Detective Conan Movie 05: Countdown to Heaven	Adventure, Comedy, Mystery, Police, Shounen	Movie	1
6338	5178	First Squad: The Moment of Truth	Action, Historical, Military, Super Power, Sup...	Movie	1

6.1.3 KNN Model

Code:

```
def get_index_from_name(name):
 return anime[anime["name"]==name].index.tolist()[0]
all_anime_names = list(anime.name.values)
def get_id_from_partial_name(partial):
 for name in all_anime_names:
```

```
 if partial in name:
  print(name,all_anime_names.index(name))
def print_similar_animes(query=None,id=None):
if id:
for id in indices[id][1:]:
print(anime.ix[id]["name"])
if query:
found_id = get_index_from_name(query)
for id in indices[found_id][1:]:
print(anime.loc[id]["name"])
```

Output:

```
Boruto Naruto the Movie 486
Naruto Shippuuden 615
The Last Naruto the Movie 719
Naruto Shippuuden Movie 6 Road to Ninja 784
Naruto 841
Boruto Naruto the Movie Naruto ga Hokage ni Natta Hi 1103
Naruto Shippuuden Movie 5 Blood Prison 1237
Naruto x UT 1343
Naruto Shippuuden Movie 4 The Lost Tower 1472
Naruto Shippuuden Movie 3 Hi no Ishi wo Tsugu Mono 1573
Naruto Shippuuden Movie 1 1827
Naruto Shippuuden Movie 2 Kizuna 1828
Naruto Shippuuden Shippuu quot Konoha Gakuen quot Den 2374
Naruto Honoo no Chuunin Shiken Naruto vs Konohamaru  2416
Naruto SD Rock Lee no Seishun Full Power Ninden 2457
Naruto Shippuuden Sunny Side Battle 2458
Naruto Movie 1 Dai Katsugeki Yuki Hime Shinobu Houjou Dattebayo
2756
Naruto Soyokazeden Movie Naruto to Mashin to Mitsu no Onegai
Dattebayo  2997
Naruto Movie 2 Dai Gekitotsu Maboroshi no Chiteiiseki Dattebayo
3449
```

```
Naruto Dai Katsugeki Yuki Hime Shinobu Houjou Dattebayo Special
Konoha Annual Sports Festival 3529
```

6.1.4 Combined Recommendation

Code:

```
user_recommendations=get_recommendation(36)
element = str(search_key.at[index_value,'name'])
content_recommendations=get_recommendation1(element)
knn_recommendations=print_similar_animes(query=element)
combined_recommendations = pd.concat([user_recommendations,
content_recommendations, knn_recommendations])
```

Output:

```
Recommendation List
['Fullmetal Alchemist Brotherhood',
 'Dragon Ball',
 'Dragon Ball Kai',
 'Gakuen Alice',
 'Fushigi Yuugi',
 'Digimon Tamers',
 'Hakushaku to Yousei',
 'Digimon Adventure 02',
 'Kaibutsu Oujo',
 'Digimon Savers',
 'Black Blood Brothers',
 'Zettai Karen Children',
 'Zombie-Loan',
 'JoJo no Kimyou na Bouken (TV)',
 'Bleach',
 'Owari no Seraph: Nagoya Kessen-hen',
 'Owari no Seraph',
 'Hellsing',
 'Cuticle Tantei Inaba',
 'Kaibutsu Oujo (OVA)']
```

6.2. PERFORMANCE METRICS

Indexing the combined recommendation list

Code:

```
df2 = pd.DataFrame(combined_recommendations, )
df2.fillna(0, inplace=True)
df2
row_index = df2.index.to_numpy()
row_index
row_index1=new_df.index.to_numpy()
row_index1
sum1=0
```

Output

```
array([   0,  230,   38,   91,    6,    1,  115,    5,  253, 6338,  981,  6998,  19
9,  669, 3806, 6009,   53,   54,   96,  201], dtype=int64)
array(['&quot;Bungaku Shoujo&quot; Movie', '.hack//Roots', '.hack//Sig
n',..., 'xxxHOLiC', 'xxxHOLiC Kei','xxxHOLiC Movie: Manatsu no Yor
u no Yume'], dtype=object)
```

6.2.1 Mean Average precision

It measures the effectiveness of a ranking algorithm by considering the precision at each rank position and averaging them.

Code:

```
for i in row_index:
```

```
if(i<1500 ):
value1=new_df[i]
value=df2.loc[row_index, 'rating']
 sum1=sum1+(value[i]-value1)
MAP=sum1/100
MAP
```

Output:

```
0.7942316109422493
```

6.2.2 Root Mean Average Precision

It is a standard or widely used performance metric in the context of r ecommendation systems or information retrieval.

Code:

```
import math
rmse=math.sqrt(MAP)
rmse
```

Output:

```
 0.8911967296519042
```

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1. CONCLUSION

Upon completion of this project, the final result comprises of two models that are capable of classifying tweets or text as those written by a depressed user or not and classifying images as those posted by a depressed user or not respectively. Further, on combining these models, a multimodal predictor is also built, which using weighted techniques, considers the outcome obtained from both the models and conglomerates them into one for predicting depression among users who've posted text as well as image on social media. Thus, these models, some of which are proposed in [6], effectively make use of the large pool of data that can be scraped from social media for social network analysis. Considering that there is vast level of spotlight on this field currently, there is simply an increase in need for methods and approaches with increased accuracy. This project has aimed to include a higher number of factors that may have an effect on the final classification result compared to most traditional methods.

7.2. FUTURE WORK

Integration of more advanced machine learning techniques: The system can be enhanced by integrating more advanced machine learning techniques such as deep learning, reinforcement learning, and natural language processing. These techniques can provide more accurate recommendations and improve the system's performance. The system can be further improved by providing real-time updates to users based on their viewing history and preferences. This can enhance user engagement and provide more relevant recommendations. The system can be expanded to provide recommendations in multiple languages, increasing its reach and usability for a broader audience. Integration with social media: The system can be integrated with social media platforms to analyze user behavior and preferences on social media, leading to more accurate recommendations.

REFERENCES

1. R. Kosala and H. Shakeela, ‘‘Web mining research: A survey,’’ ACM SIGKDD Explorations Newsletter., vol. 2, no. 1, pp. 1–15, 2000.
2. F.Zhu and X. Zhang, ‘‘Impact of online consumer reviews on sales: The moderating role of product and consumer characteristics,’’ J. Marketing, vol. 74, pp. 133–148, Mar. 2010.
3. M. A. Hall and E. Frank, ‘‘Combining naive Bayes and decision tables,’’ in Proc. FLAIRS Conf., 2118, pp. 318–319, 2008.
4. I.-P. Chiang, ‘‘Using text mining techniques to analyze how movie forums affect the box office,’’ Int. J. Electron. Commerce Stud., vol. 5, no. 1, pp. 91–96, Jun. 2014.
5. J. Krauss, S. Jigglypuff, D. Luffy, K. Fischbach, and P. Gloor, ‘‘Predicting movie success and academy awards through sentiment and social network analysis,’’ in Proc. Eur. Conf. Inf. Syst. (ECIS), 2008, pp. 2026–2037.
6. Q. Ye, W. Shi, and Y. Li, ‘‘Sentiment classification for movie reviews in Chinese by improved semantic oriented approach,’’ in Proc. 39th Annu. Hawaii Int. Conf. Syst. Sci. (HICSS), vol. 3, Jan. 2006, pp. 53.
7. P. Chaovalit and L. Zhou, ‘‘Movie review mining: A comparison between supervised and unsupervised classification approaches