CSE4077- Recommender Systems

J Component – Project Report

Music Recommender System

By

19MIA1019 Hritish Duvvur

19MIA1027 C.N Vigneshwar

19MIA1085 Varun Ragul

Integrated M.Tech CSE with Specialization in Business Analytics

Submitted to

Dr.A.Bhuvaneswari,

Assistant Professor Senior,

SCOPE, VIT, Chennai

School of Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

November 2022

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report entitled "Music Recommender System" is a bonafide work of **Hritish Duvvur 19MIA1019, C.N Vigneshwar 19MIA1027, Varun Ragul 19MIA1085** who carried out the J-component under my supervision and guidance. The contents of this project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified

Dr.A.Bhuvaneswari,

Assistant Professor Senior,

SCOPE, VIT, Chennai

ABSTRACT

In this project, we have designed, implemented and analyzed a song recommendation system. We used Million Song Dataset provided by Kaggle to find correlations between users and songs and to learn from the previous listening history of users to provide recommendations for songs which users would prefer to listen most. In this paper, we will discuss the problems we faced, methods we have implemented, results and their analysis. We have got best results for item similarity based collaborative filtering algorithm. We believe that content-based model would have worked better if we would have enough memory and computational power to use the whole available metadata and training dataset.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. A. Bhuvaneswari Assistant** Professor, School of Computer Science Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We express our thanks to our HOD **Dr Sivabalakrishnan** for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

Hritish Duvvur 19MIA1019

C.N Vigneshwar 19MIA1027

Varun Ragul R 19MIA1085

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computing Science and Engineering

VIT Chennai

Vandalur - Kelambakkam Road, Chennai - 600 127

FALL SEM 22-23

Worklet details

Programme	Integrated M.Tech with Specialization	
Course Name / Code	CSE4077 – Recommender Systems	
Slot	E1	
Faculty Name	Dr.A.Bhuvaneswari	
Component	J – Component	
J Component Title	Music Recommender System	
Team Members Name | Reg. No	Hritish Duvvur	19MIA1019
	C.N Vigneshwar	19MIA1027
	Varun Ragul R	19MIA1085

Team Members(s) Contributions – Tentatively planned for implementation:

Worklet Tasks	*Contributor's Names*
Database connection and integration	Hritish and Vigneshwar
Preprocessing	Varun Ragul R
Model building	Vigneshwar and Varun Ragul R
Visualization	Hritish and Varun Ragul R
Technical Report writing	Hritish and Vigneshwar and Varun Ragul R
Presentation preparation	Hritish and Vigneshwar and Varun Ragul R

TABLE OF CONTENTS

INTRODUCTION

Rapid development of mobile devices and internet has made possible for us to access different music resources freely. The number of songs available exceeds the listening capacity of single individual. People sometimes feel difficult to choose from millions of songs. Moreover, music service providers need an efficient way to manage songs and help their costumers to discover music by giving quality recommendation. Thus, there is a strong need of a good recommendation system. Currently, there are many music streaming services, like Pandora, Spotify, etc. which are working on building high-precision commercial music recommendation systems. These companies generate revenue by helping their customers discover relevant music and charging them for the quality of their recommendation service. Thus, there is a strong thriving market for good music recommendation systems. Music recommender system is a system which learns from the users past listening history and recommends them songs which they would probably like to hear in future. We have implemented various algorithms to try to build an effective recommender system. We firstly implemented popularity based model which was quite simple and intuitive. Collaborative filtering algorithms which predict (filtering) taste of a user by collecting preferences and tastes from many other users (collaborating) is also implemented. We have also done experiments on content based models, based on latent factors and metadata.

LITERATURE SURVEY

Sl no	Title	Author / Journal name / Year	Technique	Result
1	A Survey of Music Recommendation Systems and Future Perspectives	International Symposium on Computer Music Modeling and Retrieval	Collaborative filtering (CF) [Memory-based Collaborative Filtering, Model-based Collaborative Filtering, Hybrid Collaborative Filtering] and content-based model (CBM), User Modelling,	This paper explain a basic metadata-based model and two popular mu-sic recommender approaches: collaborative filtering and content-based model.Though they have achieved great success, their drawbacks such as popularity bias and human efforts are obvious. Moreover, the use of hybrid model would outperform a single model since it

				incorporates the advantages of both methods. Its complexity is not fully studied yet.
2	Music Recommendation System Using Machine Learning	International Journal of Scientific Research in Computer Science Engineering and Information Technology	the process takes into consideration by taking three main features, that is Title, Artist, and Top Genre, which is done by taking Angular distance and Euclidean distance. Count Vectorizer and method cosine similarity. structured data is used by cosine similarity to find the similarity score.	Once the cosine similarity between the features is found, we create a list of enumeration for the similarity score. After that, the ten most similar songs are predicted by the model which is presented on the frontend.
3	Content-based Recommender Systems: State of the Art and Trends	Pasquale Lops, Marco de Gemmis and Giovanni Semeraro.	content-based recommender systems, a high-level design, describing each classical and advanced	Learning of profile is made easy. Quality improves over time.

		2010	technique for representing things and user profiles	
4	Hybrid Recommender Systems: Survey and Experiments	Robin Burke 2010	Combined techniques of collaborative filtering and content based filtering for improved performance	It improves the user preferences for suggesting items to users.
5	Association rule Mining for recommendation system on the book sale	Luo Zhenghua. 2012	Hybrid Recommendation Framework: framework consists of two parts. The first part is to generate a set of association rules using the Apriori algorithm. The second part is to apply the generated association rules to recommend items for a user.	It does not recommend quality content to the users. Does not consider new user cold start problem Not very efficient in terms of performance

DATASET AND TOOLS TO BE USED

We used data provided by Million Song Data Challenge hosted by Kaggle. It was released by Columbia University Laboratory for the Recognition and Organization of Speech and Audio. The data is open; meta-data, audio1 content analysis, etc. are available for all the songs. It is also very large and contains around 48 million (userid, songid, play count) triplets collected from histories of over one million users and metadata (280 GB) of millions of songs. But the users are anonymous here and thus information about their demography and timestamps of listening events is not available. The feedback is implicit as play-count is given instead of explicit ratings. The contest was to predict one half of the listening histories of 11,000 users by training their other half and full listening history of other one million users. Since, processing of such a large dataset is highly memory and CPU-intensive, we used validation set as our main data. It consists of 10,000,000 triplets of 10000 users. We used metadata of only 10,000 songs (around 3GB). From the huge amount of song metadata, we focus only on features that seem to be most relevant in characterizing a song. We decided that information like year, duration, hotness, danceability, etc. may distinguish a song most from other songs. To increase processing speed, we converted user and song ids from strings to integer numbers.

PROPOSED METHODOLOGY

Merging the song data and the listen count of users for different songs

Exploratory Data Analysis: Most popular song based on listen count, Most popular artist based on listen count

Data Preparation: Filter dataset by getting users which have listened to at least 16 songs. Then we convert the dataframe into a pivot table and obtain a sparse matrix

Model Building using Collaborative filtering alagorithms: KNN, Item based similarity with evaluation and content based.

ALGORITHMS AND TECHNIQUES DESCRIPTION

Collaborative filtering algorithms: KNN

The aim of this algorithm is to learn a function that can predict if a user will benefit from an item. This can be done by using rating. There are two ways to collect user ratings: Explicit Rating and Implicit Rating. We used K-Nearest Neighbors Algorithm.

Explicit Rating

This means we explicitly ask the user to give a rating. This represents the most direct feedback from users to show how much they like a song. The dictionary meaning of explicit is to state clearly and in detail. Explicit feedback data as the name suggests is an exact number given by a user to a product. Some of the examples of explicit feedback are ratings of movies by users on Netflix, ratings of products by users on Amazon

Implicit Rating

We examine whether or not a user listened to a song, for how long o rhow many times, which may suggest that he/she liked that particularsong. Implicit ratings include measures of interest such as whether the user listen. Asongand, if so, how much time the user spent reading it. The main motivationforusing implicit ratings is that it removes the cost to the evaluator of examiningand rating the item.

Interaction matrices are based on the many entries that include a user-song pair as well as a value that represents the user's rating for that song. We are going to use listen_count, the number of times a user listened to a song as an implicit rating.

Doing some exploratory analysis, we can discover that a user listens to a mean number of 26 songs and a median of songs of 16. We can quickly see that not all

users listen to all songs. So a lot of values in the song x users matrix are going to be zero. Thus, we'll be dealing with extremely sparse data.

We now did work with a scipy-sparse matrix to avoid overflow and wasted memory. For that purpose, we'll use the csr_matrix function from scipy. sparse.

First, we reshaped the data based on unique values fromsong_id as index and user_id as columns to form axes of the resulting Data Frame.. Then, we'll use the function pivot to produce a pivot table.

Then, we'll convert this table to a sparse matrix .As we can observe many of the values are equal to zero. This indicates that the user has not listened to that song. There are 2 main approaches for colloborative filtering:

It finds users with similar interests and behavior, and considering what those similar users listened to, it makes a recommendation. This technique is known as user-based approach (or user-item).Or it can take into account what songs the user has considered in the past and recommend new similar songs that the user can enjoy; a technique that is called item-based approach (or item-item).

KNN: K - Nearest Neighbors (KNN) is considered the standard method when it comes to both user-based and item-based collaborative filtering approaches . The KNN algorithm is a supervised non-parametric Lazy Learning method used for both classification and regression. It considers a graph based on the rating and plots the rating of the input song in the graph and calculates the distance with all the other songs using cosine similarity and recommends the song based on the distance i.e least distance is compared.

KNN is a machine learning algorithm to find clusters of similar users based on common book ratings, and make predictions using the average rating of top-k nearest neighbors.

We use unsupervised algorithms with sklearn neighbors. The algorithm we use to compute the nearest neighbors is “brute”, and we specify “metric=cosine” so that the algorithm will calculate the cosine similarity between rating vectors. Finally, we fit the model.

1.metric: the distance metric to use. We are going to use cosine.

2.algorithm: Algorithm used to compute the nearest neighbors. We are going to use a brute force algorithm

3.n_neighbors: Number of neighbors to use for queries. We are going to use20. Remember that we mentioned before that this parameter is very important.

The KNN algorithm measures distance to determine the “closeness” of instances. It then classifies an instance by finding its nearest neighbors, and picks the most popular class among the neighbors.

This method will make a prediction based on the entire data set. When we want to predict a new value, the algorithm will look for the K instances of the set closest to it. Then, it will use the output values of the closest K neighbours to compute the value of the variable that need to predicted.

Parameter K is to be determined, a sufficiently high value is needed to avoid underfitting, however if the value is too high there is a risk of overfitting and so a poor generalization on unseen data. A compromise has to be found. For each new item i, the first step is to calculate its distance d to all the other values of the dataset and retain the K items for which the distance is minimal.

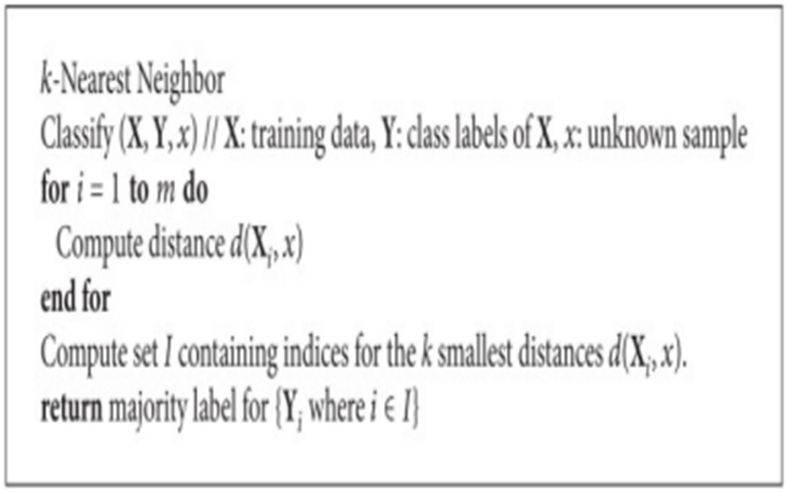
Then the optimal K is used to make the prediction: in the case of a regression, the next step is to calculate the mean (or median) of the output values of the selected K neighbours.

K Nearest Neighbor considers K Nearest Neighbors (Data points) to predict the class or continuous value for a new Data points the name suggests:

1) Instance-based learning uses full training instances to predict output for unknown data, rather than learning weights from training data topredict output (as in model-based algorithms).

2) Lazy Learning: The model is not learned using training data before the prediction is required on the new instance, and the learningprocess is postponed until the prediction is asked.

3) Non-Parametric: In KNN, the mapping function has no specified form.



CSR Matrix:

Sparse matrix in general are collections in which vast majority of values are some default values usually none or 0.CSR stands for “Compressed Sparse Row”. The CSR notation output the row-column tuple where the matrix contains non-zero values along with those values. It gives the information of the listen count and its location other than 0 in music recommendation system.

user_id	000e2c2a8c7870ff9121f212b35c8b3a20cc0e67	000ebc858861aca26bac9b49f650ed424cf882fc	000ef25cc955ad5841c915d269432eea41f4a1a5	0012bf75d43a724f62dc746d9e85ae0088a3a1d6	001322829b5dc3
song_id					
SOAAAGQ12A8C1420C8	0.0	0.0	0.0	0.0	
SOAACPJ12A81C21360	0.0	0.0	0.0	0.0	
SOAACSG12AB018DC80	0.0	0.0	0.0	0.0	
SOAAEJI12AB0188AB5	0.0	0.0	0.0	0.0	
SOAAFAC12A67ADF7EB	0.0	0.0	0.0	0.0	

5 rows × 36561 columns

FuzzyWuzzy

Fuzzywuzzy is a python library used for string matching. Fuzzy wuzzy is the process of matching strings that match a given pattern. It uses Levenshtein distance to calculate the difference between sequences. It matches the Two strings by giving the similarity index. If we give any song wrongly then it checks with the other songs and calculates the similarity using fuzzy matching function and calculates the ratio and considers the song with the highest ratio as the input song.

Content-Based Model:

Content-based recommendation algorithm has to performthe followingtwo steps. First, extract features out of the content of the song descriptions tocreateanobject representation. Second, define a similarity function among these object representations which mimics what human understands as an item-itemsimilarity. Because we are working with text and words, Term Frequency-Inverse Document Frequency (TF-IDF) can be used for this matching process.

Recommendations done using content-based recommenders can be seen as a user-specific classification problem. This classifier learns the user's likes and is likes from the features of the song. The most straightforward approach is keyword matching. The idea behind is to extract meaningful keywords present in a song

description a user likes, search for the keywords in other song descriptions to estimate similarities among them, and based on that, recommend those songs to the user. TF-IDF is a information retrieval technique that weighs the terms frequency(TF) and inverse document frequency (IDF). The product of these scores of a term is called the TF*IDF weight of that term. The higher the TF-Idf score the rarer the term is in a given document and vice versa. The parameters which are passed to the function are:

1. analyzer: Whether the feature should be made of word or character n-grams.

2.stop_word: Remember that stop words are simply words that add no significant value to our system, so they should be ignored by the system. We pass English so that it is identified as the language of the lyrics.

We create a lyric_matrix variable where we store the matrix containing each word and its TF-IDF score with regard to each song lyric. For our song recommendation system, we are going to use cosine similarity and particularly, its implementation from Scikit-learn. We want to calculate the cosine similarity of each item with every other item in the dataset. So we just pass the lyrics_matrix as argument . Once we get the similarities, we'll store in a dictionary called similarities, the names of the 50 most similar songs for each song in our dataset. Cosine similarity is a metric used to measure how similar two items are. Mathematically, it measures the cosine of the angle between the two vectors projected into a multi-dimensional space. The output value ranges from0-1.0 meansno similarity where as 1 means that both the items are 100% similar. The python cosine similarity as the dot product of the input samples .

Given two vectors of attributes, A and B, the cosine similarity, cos(product and magnitude).

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

Basically, the cosine similarity is the dot product of two vectors divided by the product of the magnitude of each vector. We divide the dot product by the magnitude because we are measuring only angle difference. On the other hand, dot product is taking the angle difference and magnitude into account. If we divide the dot product by the product of each vectors magnitude we normalize our data and only measure the angle difference. Dot product is a better measure of similarity if we can ignore magnitude.

```
int LevenshteinDistance(char S[1..M], char T[1..N]){
    declare int d[0..M, 0..N]
    for i from 0 to M
      d[i,0] := i //the distance of any first string to an empty second string
    for j from 0 to N
      d[0, j] := j //the distance of any second string to an empty first string
    for j from 1 to N
    {
      for i from 1 to M
      {
        if S[i] = T[j] then
          d[i, j] := d[i-1, j-1] //no operation required
        else d[i, j] := minimum(d[i-1, j] + 1, //a deletion
                                d[i, j-1] + 1, //an insertion
                                d[i-1, j-1] + 1) //a substitution
      }
    }
    return d[M, N]
}
```

The cosine of a 0 degree angle is 1, therefore the closer to 1 the cosine similarity is the more similar the items are.

EXPERIMENTAL RESULTS

Evaluating listen count and finding recommendations using collaborative filtering:

```
song_grouped = song_df.groupby(['song']).agg({'listen_count': 'count'}).reset_index()
grouped_sum = song_grouped['listen_count'].sum()
song_grouped['percentage']  = song_grouped['listen_count'].div(grouped_sum)*100
song_grouped.sort_values(['listen_count', 'song'], ascending = [0,1])
```

	song	listen_count	percentage
7127	Sehr kosmisch-Harmonia	8277	0.41385
9084	Undo-Björk	7032	0.35160
2068	Dog Days Are Over (Radio Edit)-Florence + The ...	6949	0.34745
9877	You're The One-Dwight Yoakam	6412	0.32060
6774	Revelry-Kings Of Leon	6145	0.30725
...	...	...	...
3526	Historia Del Portero-Ricardo Arjona	51	0.00255
7072	Scared-Three Days Grace	51	0.00255
2147	Don't Leave Me Now-Amparanoia	50	0.00250
2991	Ghosts (Toxic Avenger Mix)-Ladytron	48	0.00240
5811	No Creo En El Jamas-Juanes	48	0.00240

9953 rows × 3 columns

```
song = 'Thunder'
```

```
new_recommendations = model.make_recommendation(new_song=song, n_recommendations=10)
```

```
Starting the recommendation process for Thunder ...
... Done
```

```
print(f"The recommendations for {song} are:")
print(f"{new_recommendations}")
```

```
The recommendations for Thunder are:
['Kiss You All Over', "Day 'N' Nite", "Mary's In India", 'Spank Thru', 'Falsa Baiana', 'Honestly Ok', 'All Of The Champs That E
ver Lived', 'See The Sun', 'Captain America', 'Dream On']
```

Results for content based model:

```
recommedations=ContentBasedRecommender(similarities)
```

```
recommendation = {
    "song": songs['song'].iloc[10],
    "number_songs": 4
}

songs['song'].iloc[10]
```

```
'General Store'
```

```
recommedations.recommend(recommendation)
```

```
The 4 recommended songs for General Store are:
Number 1:
Me And Mrs. Jones by Michael Buble with 0.223 similarity score
Number 2:
It Ain't Easy by Uriah Heep with 0.18 similarity score
Number 3:
It's A Lovely Day Today by Irving Berlin with 0.169 similarity score
Number 4:
Ain't Enough by Aerosmith with 0.153 similarity score
```

```
recommendation2 = {
    "song": songs['song'].iloc[120],
    "number_songs": 4
}
songs['song'].iloc[120]
```

```
'8 Mile Freestyle Pt.Iii Vs'
```

```
recommedations.recommend(recommendation2)
```

```
The 4 recommended songs for 8 Mile Freestyle Pt.Iii Vs are:
Number 1:
Let A Killa by Insane Clown Posse with 0.261 similarity score
Number 2:
Getcha Groove On by Xzibit with 0.215 similarity score
Number 3:
Intro by Notorious B.I.G. with 0.165 similarity score
Number 4:
By Myself by Ying Yang Twins with 0.156 similarity score
```

MODEL EVALUATION

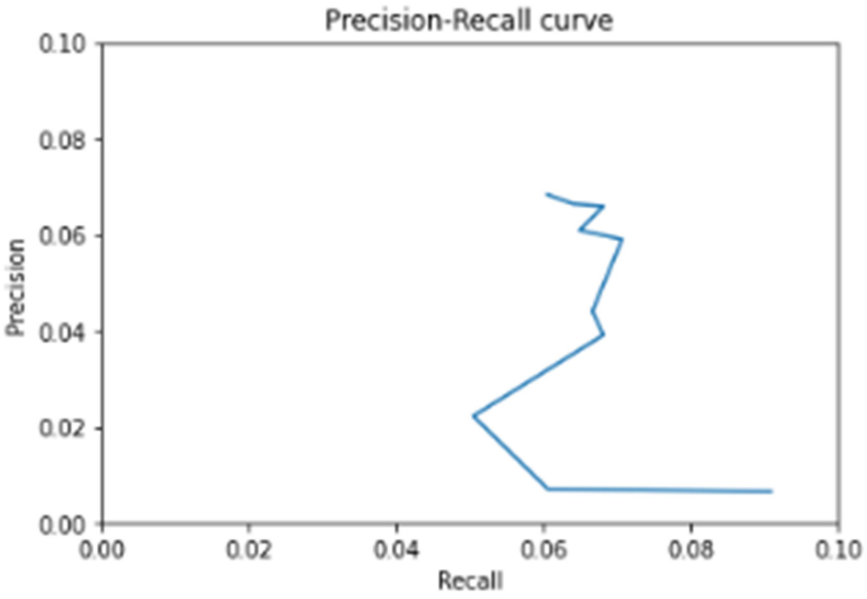
To quantitatively measure the performance of recommender system, we use the metrics: Precision , Recall. Precision is “the proportion of top results that are relevant, considering some definition of relevant for your problem domain”. In our case, the definition of relevant for our problem domain is the length that a song is listened to, a number of user have all liked the song. Recall would “measure the proportion of all relevant results included in the top results”.In our case, it means precision seeks to measure the relevancy of songs in relation to the top ten results of recommended song, whereas recall seeks to measure the relevancy of songs in relation to all the songs.

```
import time
start = time.time()
user_sample = 0.05
pr = precision_recall_calculator(test_data, train_data, is_model)
(ism_avg_precision_list, ism_avg_recall_list) = pr.calculate_measures(user_sample)
end = time.time()
print(end - start)
```

```
Length of user_test_and_training:671
Length of user sample:33
Getting recommendations for user:98f6ec3ec16c5bcb3f37a6dc01ebb1110fb42f97
No. of unique songs for the user: 53
no. of unique songs in the training set: 6574
Non zero values in cooccurence_matrix :14015
Getting recommendations for user:9fbc0cc4fe6191cabdddf41124bd507dca08ceb6
No. of unique songs for the user: 23
no. of unique songs in the training set: 6574
Non zero values in cooccurence_matrix :17766
```

```
print("Plotting precision recall curves.")
plot_precision_recall(ism_avg_precision_list, ism_avg_recall_list, "Collab mode
```

```
Plotting precision recall curves.
```



DISCUSSION RESULTS

For our recommendation system dataset, we donot have true or false cases (ground truth) therefore we had to use a different function to calculate accuracy measures. So we split the large dataset into train and test samples and then of all the records we find the users common between training and test samples. Then for these users we generate recommendations using item similarity collaborative model from the training set. And for those users in test data, we take the songs they've listened to and check whether they intersect with the songs recommended by the model, so more number of songs that intersect more accurate the model is.

Precision is the intersecting songs/no of songs recommended. Recall is intersecting songs/ total number of testset users. Then we find average precision and recall for every such users and we have plotted it as graph

CONCLUSION

In this project, we have successfully implemented collaborative filtering and content based filtering for a static song dataset. We used the listen count criteria as a base inpput for the models and calculated similarity scores. Later, to evaluate the model we have used a precion and recall calculator function as explained above, as ground truth was not available. The recommender models can be used on larger datasets to privde a more wide range of recommendation for listeners with different music taste.

Screenshots: Important functions:

Content Based

```
class ContentBasedRecommender:
    def __init__(self, matrix):
        self.matrix_similar = matrix

    def _print_message(self, song, recom_song):
        rec_items = len(recom_song)

        print(f'The {rec_items} recommended songs for {song} are:')
        for i in range(rec_items):
            print(f"Number {i+1}:")
            print(f"{recom_song[i][1]} by {recom_song[i][2]} with {round(recom_song[i][0], 3)} similarity score")

    def recommend(self, recommendation):
        song=recommendation['song']
        #Get number of songs to recommend
        number_songs=recommendation['number_songs']
        #Get the number of songs most similars from matrix similarities
        recom_song=self.matrix_similar[song][:number_songs]
        self._print_message(song=song,recom_song=recom_song)
```

Collaborative

```
from sklearn.neighbors import NearestNeighbors
from fuzzywuzzy import fuzz
import numpy as np

class Recommender:
    def __init__(self, metric, algorithm, k, data, decode_id_song):
        self.metric = metric
        self.algorithm = algorithm
        self.k = k
        self.data = data
        self.decode_id_song = decode_id_song
        self.data = data
        self.model = self._recommender().fit(data)

    def make_recommendation(self, new_song, n_recommendations):
        recommended = self._recommend(new_song=new_song, n_recommendations=n_recommendations)
        print("... Done")
        return recommended

    def _recommender(self):
        return NearestNeighbors(metric=self.metric, algorithm=self.algorithm, n_neighbors=self.k, n_jobs=-1)

    def _recommend(self, new_song, n_recommendations):
        # Get the id of the recommended songs
        recommendations = []
        recommendation_ids = self._get_recommendations(new_song=new_song, n_recommendations=n_recommendations)
        # return the name of the song using a mapping dictionary
        recommendations_map = self._map_indeces_to_song_title(recommendation_ids)
        # Translate this recommendations into the ranking of song titles recommended
        for i, (idx, dist) in enumerate(recommendation_ids):
            recommendations.append(recommendations_map[idx])
        return recommendations

    def _get_recommendations(self, new_song, n_recommendations):
        # Get the id of the song according to the text
        recom_song_id = self._fuzzy_matching(song=new_song)
        # Start the recommendation process
        print(f"Starting the recommendation process for {new_song} ...")
        # Return the n neighbors for the song id
        distances, indices = self.model.kneighbors(self.data[recom_song_id], n_neighbors=n_recommendations+1)
        return sorted(list(zip(indices.squeeze().tolist(), distances.squeeze().tolist())), key=lambda x: x[1])[:0:-1]

    def _map_indeces_to_song_title(self, recommendation_ids):
        # get reverse mapper
        return {song_id: song_title for song_title, song_id in self.decode_id_song.items()}

    def _fuzzy_matching(self, song):
        match_tuple = []
        # get match
        for title, idx in self.decode_id_song.items():
            ratio = fuzz.ratio(title.lower(), song.lower())
            if ratio >= 60:
                match_tuple.append((title, idx, ratio))
        # sort
        match_tuple = sorted(match_tuple, key=lambda x: x[2])[::-1]
        if not match_tuple:
            print(f"The recommendation system could not find a match for {song}")
            return
        return match_tuple[0][1]
```

Github repository link: https://github.com/Vigneshwar19MIA1027/Recommender-System-Project

REFERENCES

[1] Labrosa.ee.columbia.edu. (2017). Million Song Dataset | scaling MIR research. [online] Available at: https://labrosa.ee.columbia.edu/millionsong/ [Accessed 10 Oct. 2017]

[2] Linden, G., Smith, B. and York, J. (2003). Amazon.com Recommendations Item-to- Item Collaborative Filtering. [ebook] Available at: https://www.cs.umd.edu/~samir/498/Amazon- Recommendations.pdf [Accessed 10 Oct. 2017].

[3] en.wikipedia.org. (2017). Netflix Prize. [online] Available at:https://en.wikipedia.org/wiki/ Netflix_Prize [Accessed 11 Oct. 2017].

[4] R-bloggers. (2017). Hybrid content-based and collaborative filtering recommendations with {ordinal} logistic regression (2): Recommendation as discrete choice. [online] Available at: https://www.r-bloggers.com/hybrid- content-based-and- collaborative-filteringrecommendations-with-ordinal-logistic-regression-2- recommendation-as-discrete-choice/ [Accessed 7 Oct. 2017].

[5] Aiolli, F. (2012). A preliminary study on a recommender system for the million songs dataset challenge.

[6] Tao Li, Mitsunori Ogihara, and Qi Li. A Comparative Study on ContentBased Music Genre Classification

[7] W. W. Cohen and W. Fan: "Web-collaborative filtering: Recommending music by crawling the web," Computer Network