Mini Project

On

SONG RECOMMENDATION SYSTEM

Submitted in partial fulfilment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

BY

17WH1A0551	**B.PRANEETHA**
17WH1A0525	**K.PRAVALLIKA**
17WH1A0516	**S.SAMIKSHA**

Under the esteemed guidance of

Mr U. Chandrasekhar

Associate Professor

Department of Computer Science & Engineering

BVRIT HYDERABAD

COLLEGE OF ENGINEERING FOR WOMEN

(NBA Accredited EEE.ECE.CSE.IT B.Tech Courses)

(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

January, 2021

BVRIT HYDERABAD

COLLEGE OF ENGINEERING FOR WOMEN

(NBA Accredited EEE.ECE.CSE.IT B.Tech Courses)

(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

Department of Computer Science & Engineering

CERTIFICATE

This is to certify that the mini project entitl**ed "Song Recommendation System**" is a bonafide work carried out by **Ms. B.Praneetha(17wh1a0551) , Ms.K.Pravallika(17wh1a0525), Ms. S. Samiksha(17wh1a0516)** in partial fulfilment for the award of B.Tech degree in **Computer Science & Engineering** , **BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad** affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision. The results embodied in the project work have not been submitted to any this University or Institute for the award of any degree or diploma.

Internal Guide	**Head of the Department**
Mr U. Chandrasekhar	**Dr. Srinivasa Reddy Konda**
Associate Professor, CSE	**Professor, CSE**

External Examiner

DECLARATION

We hereby declare that the work presented in this project entitled **"Song Recommendation System"** submitted towards completion of Project work in IV Year of B.Tech of CSE at **BVRIT HYDERABAD College of Engineering for Women,** Hyderabad is an authentic record of our original work carried out under the guidance of **Mr U. Chandrasekhar , Associate Professor, Department of CSE.**

Sign with Date:
B.Praneetha
(17wh1a0551)

Sign with Date:
K.Pravallika
(17wh1a0525)

Sign with Date:
S. Samiksha
(17wh1a0516)

ACKNOWLEDGEMENT

We would like to express our sincere thanks to **Dr.K.V.N. Sunitha**, **Principal, BVRIT HYDERABAD College of Engineering for Women,** for his support by providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr. Srinivasa Reddy Konda, Head, Department of CSE, BVRIT HYDERABAD College of Engineering for Women,** for all timely support and valuable suggestions during the period of our project.

We are extremely thankful to our Internal Guide and Mini Project Coordinator, **Mr U. Chandrasekhar, Associate Professor, CSE, BVRIT HYDERABAD College of Engineering for Women,** for his constant guidance and encouragement throughout the project.

Finally, we would like to thank all Faculty and Staff of CSE department who helped us directly or indirectly. Last but not least, we wish to acknowledge our **Parents** and **Friends** for giving moral strength and constant encouragement.

Ms. B. Praneetha (17wh1a0551)

Ms. K. Pravallika (17wh1a0525)

Ms. S. Samiksha (17wh1a0516)

ABSTRACT

From past years, Machine learning is playing a major role in image detection, product recommendation, spam reorganization and medical diagnosis. In this project, a song recommendation system is designed, implemented and analysed using machine learning. Million Song Dataset provided by Kaggle is used to find correlations between users and songs to provide recommendations for songs which users would prefer to listen. In this work problems faced, methods implemented, results and their analysis will be discussed. Memory based collaborative filtering algorithm gave best results although content-based model would have worked better if had enough memory and computational power to use the whole available metadata and training dataset.

LIST OF FIGURES

S.No	Figure description	Page no.
1	Flow chart	10
2	Importing	11
3	Loading directories	11
4	Processing data	12
5	Model training	12
6	Recommendation of songs	13
7	Matrix M	16
8	Result	19

LIST OF CONTENTS

S.No	Topic	Page no.
1.	Introduction	9
1.1	Problem statement	9
1.2	Objective	9
2	Requirement	9
2.1	Software	9
2.2	Hardware	9
3	Design	10
4	Dataset	10
5	Implementation	11
5.1	Importing necessary libraries	11
5.2	Loading train, testing data and model	11
5.3	Processing data	12
5.4	Training the model	12
5.5	Function for recommending songs	12
6	Testing	18
7	Results	18
8	Conclusion	19
9	Future work	19
10	References	19

1. INTRODUCTION

1.1) PROBLEM STATEMENT

In today's world 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 interest of an individual. People sometimes feel difficult to choose from millions of songs available. Moreover, music service providers need an efficient way to manage songs and help their costumers by giving quality recommendations. Thus, there is a strong need of a good recommendation system.

Currently, these are many music streaming services, like Wynk, Spotify, etc. which are working on building good and commercial song recommendation systems. These companies generate revenue by helping their customers find relevant song and charge them. Thus, these is a strong emerging market for good song recommendation systems.

Song recommender system is a system which learns from the users past listening history and recommends them songs which they would probably like to hear. Various algorithms have been implemented to build an effective recommender system. Firstly, popularity based model which was quite simple was implemented. Collaborative filtering algorithms which predict taste of a user by collecting preferences and tastes from many other users (collaborating) is also implemented.

1.2) OBJECTIVE

This project attempts to address the question by using machine learning techniques to recommend songs based on history of user.

2. REQUIREMENT

2.1)SOFTWARE

Python 3.8

Google colab

2.2)HARDWARE

Intel Core i5 Processor

RAM 8GB, Hard Disk Drive 1TB

3. DESIGN

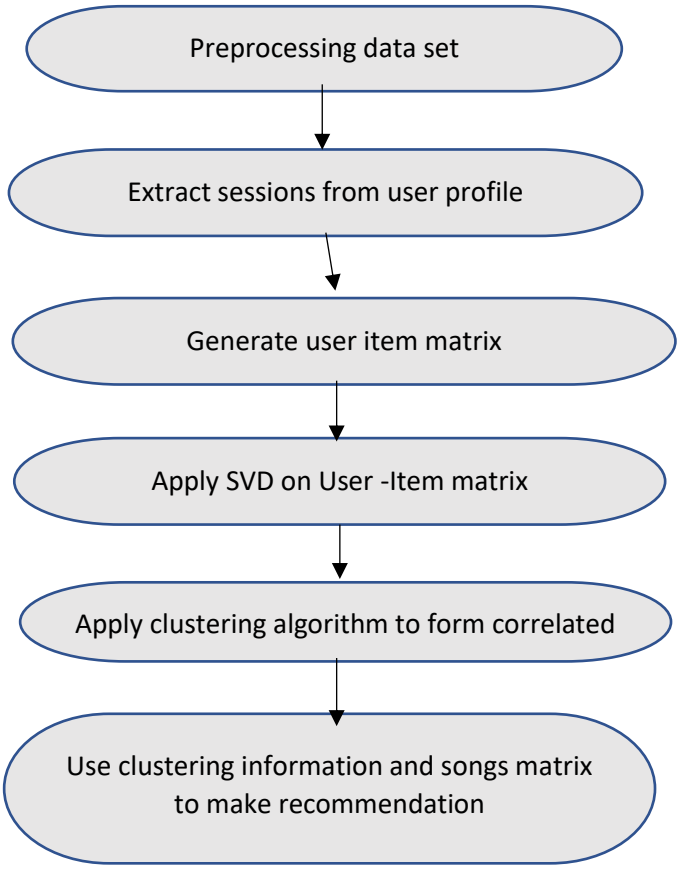


Figure 1: Flow graph

4. DATASET

Data provided by Million Song Data Challenge hosted by Kaggle is used for this project. It was released by Columbia University Laboratory for the Recognition and Organization of Speech and Audio. The data is open. 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[1]. But the users are anonymous thus information about their timestamps of listening events is not available. Since, processing of such a large dataset is highly memory and CPU intensive, validation set as our main data is used. It consists of 10,000,000 triplets of 10000 users. Metadata of only 10,000 songs (around 3GB) is used. From the huge amount of song metadata, features that seem to be most relevant in characterizing a song and that help in making recommendations is focused.

5. IMPLEMENTATION

5.1) IMPORTING NECESSARY LIBRARIES

In this step necessary libraries are imported. Numpy is used for mathematical operations for multi-dimensional arrays. It is an important library which supports large arrays and its functions. Pandas is another important library which is used for data cleaning and analysis. Scipy is another these library which is used to perform linear algebra.

```
import pandas as pd
import numpy as np
import time
from sklearn.model_selection import train_test_split
from scipy.sparse import coo_matrix
import math as mt
from scipy.sparse.linalg import * #used for matrix multiplication
from scipy.sparse.linalg import svds
from scipy.sparse import csc_matrix
from scipy.stats import skew, norm, probplot
import seaborn as sns
```

Figure 2: importing

5.2) LOADING TRAIN AND TESTING DATA AND MODEL

In this step the train data and testing data directories are loaded and the model name is initialized for further processing

```
triplets_file = 'https://static.turi.com/datasets/millionsong/10000.txt'
songs_metadata_file= 'https://static.turi.com/datasets/millionsong/song_data.csv'
```

Figure 3 : Loading directories

5.3) PROCESSING TEST DATA

In this step test data is processed and loaded for testing the model.

```
[ ] count_play_df = pd.read_table(triplets_file,header=None)
    count_play_df.columns = ['user', 'song', 'play_count']
```

```
[ ] track_metadata_df = pd.read_csv(songs_metadata_file)
```

```
unique_track_metadata_df = track_metadata_df.groupby('song_id').max().reset_index()

print('Number of rows after unique song Id treatment:', unique_track_metadata_df.shape[0])
print('Number of unique songs:', len(unique_track_metadata_df.song_id.unique()))
display(unique_track_metadata_df.head())
```

```
Number of rows after unique song Id treatment: 999056
Number of unique songs: 999056
```

	song_id	title	release	artist_name	year
0	SOAAABI12A8C13615F	Afro Jazziac	To Birdland And Hurry	Herbie Mann	2000
1	SOAAABT12AC46860F0	Herre Gud Ditt Dyre Namn Og Ære	Som Den Gyldne Sol Frembryter	Bergen Big Band	0
2	SOAAABX12A8C13FEB2	N.Y.C. Remix	Paris Can´t Wait	Guardner	0
3	SOAAACR12A58A79456	Irresistible	Wowie Zowie	Superchumbo	2002
4	SOAAACY12A58A79663	Untitled 1	Pine Cone Temples	Thuja	0

```
user_song_list_count = pd.merge(count_play_df,
                                unique_track_metadata_df, how='left',
                                left_on='song',
                                right_on='song_id')
user_song_list_count.rename(columns={'play_count':'listen_count'},inplace=True)
del(user_song_list_count['song_id'])
```

Figure 4 : Processing data

5.4) TRAINING THE MODEL

We used popularity based recommender class as a black box to train our model. We create an instance of popularity based recommender model and feed with our training data.

```
def __init__(self):
    self.train_data = None
    self.user_id = None
    self.item_id = None
    self.popularity_recommendations = None
def create(self, train_data, user_id, item_id):
    self.train_data = train_data
    self.user_id = user_id
    self.item_id = item_id

    train_data_grouped = train_data.groupby([self.item_id]).agg({self.user_id: 'count'}).reset_index()
    train_data_grouped.rename(columns = {'user_id': 'score'},inplace=True)
    train_data_sort = train_data_grouped.sort_values(['score', self.item_id], ascending = [0,1])
    train_data_sort['Rank'] = train_data_sort['score'].rank(ascending=0, method='first')
    self.popularity_recommendations = train_data_sort.head(10)
```

Figure 5 : Model training

5.5) FUNCTIOM FOR RECOMMENDING SONGS

We created a function recommend which takes in user id and returns the suitable recommendations for the user.

```
def recommend(self, user_id):
    user_recommendations = self.popularity_recommendations
    user_recommendations['user_id'] = user_id
    cols = user_recommendations.columns.tolist()
    cols = cols[-1:] + cols[:-1]
    user_recommendations = user_recommendations[cols]
    return user_recommendations
```

Figure 6 : Recommendation of songs

ALGORITHM

We have implemented four different algorithms to build an efficient recommendation system.

POPULATIRY BASED MODEL

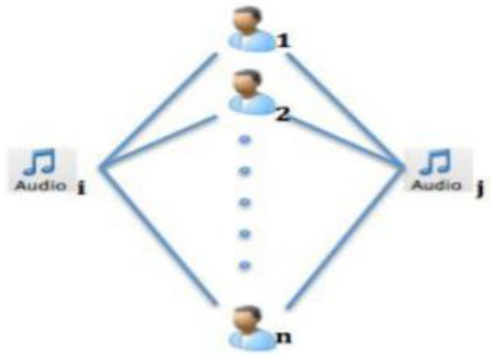
It is the most basic and simple algorithm. We find the popularity of each song by looking into the training set and calculating the number of users who had listened to this song. Songs are then sorted in the descending order of their popularity. For each user, we recommend top most popular songs except those already in his profile. This method involves no personalization and some songs may never be listened in future.

```
def create_popularity_recommendation(train_data, user_id, item_id, n=10):
    #Get a count of user_ids for each unique song as recommendation score
    train_data_grouped = train_data.groupby([item_id]).agg({user_id: 'count'}).reset_index()
    train_data_grouped.rename(columns = {user_id: 'score'},inplace=True)

    #Sort the songs based upon recommendation score
    train_data_sort = train_data_grouped.sort_values(['score', item_id], ascending = [0,1])

    #Generate a recommendation rank based upon score
    train_data_sort['Rank'] = train_data_sort.score.rank(ascending=0, method='first')

    #Get the top n recommendations
    popularity_recommendations = train_data_sort.head(n)
    return popularity_recommendations
```

COLLABORATIVE BASED MODEL

Collaborative filtering involves collecting information from many users and then making predictions based on some similarity measures between users and between items. This can be classified into user-based and item-based models.

```
def recommend(self, user):
    user_songs = self.get_user_items(user)
    print("No. of unique songs for the user: %d" % len(user_songs))
    all_songs = self.get_all_items_train_data()
    print("no. of unique songs in the training set: %d" % len(all_songs))
    cooccurence_matrix = self.construct_cooccurence_matrix(user_songs, all_songs)
    df_recommendations = self.generate_top_recommendations(user, cooccurence_matrix, all_songs, user_songs)
    return df_recommendations
def get_similar_items(self, item_list):
    user_songs = item_list
    all_songs = self.get_all_items_train_data()
    print("no. of unique songs in the training set: %d" % len(all_songs))
    cooccurence_matrix = self.construct_cooccurence_matrix(user_songs, all_songs)
    user = ""
    df_recommendations = self.generate_top_recommendations(user, cooccurence_matrix, all_songs, user_songs)
    return df_recommendations
```



In item-based model, it is assumed that songs that are often listened together by some users tend to be similar and are more likely to be listened together in future also by some other user.

According to user based similarity model, users who have similar listening histories, i.e., have listened to the same songs in the past tend to have similar interests and will probably listen to the same songs in future too.

We need some similarity measure to compare between two songs or between two users. Cosine similarity weighs each of the users equally which is usually not the case. User should be weighted less if he has shown interests to many variety of items (it shows that either she/he does not discern between songs based on their quality, or just likes to explore). Likewise, user is weighted more if listens to very limited set of songs. The similarity measure, also has drawbacks that some songs which are listened more by users have higher similarity values not because they are similar and listened together but because they are more popular.

We have used conditional probability based model of similarity between users and between items

Different values of α were tested to finally come with a good similarity measure. Then, for each new user u and song i, user-based scoring function is calculated.

Locality of scoring function is also necessary to emphasize items that are more similar. We have used exponential function to determine locality.

This determines how individual scoring components affects the overall scoring between two items. The similar things are emphasized more while less similar ones contribution drop down to zero.

After computing user-based and item-based lists, we used *stochastic* aggregation to combine them. This is done by randomly choosing one of them according to their probability distribution and then recommending top scored items from them. When the song history of a user is too small to utilize the user-based recommendation algorithm, we can offer recommendations based on song similarity, which yields better results when number of songs is smaller than that of users.

```
def construct_cooccurence_matrix(self, user_songs, all_songs):
    user_songs_users = []
    for i in range(0, len(user_songs)):
        user_songs_users.append(self.get_item_users(user_songs[i]))
    cooccurence_matrix = np.matrix(np.zeros(shape=(len(user_songs), len(all_songs))), float)
    for i in range(0,len(all_songs)):
        songs_i_data = self.train_data[self.train_data[self.item_id] == all_songs[i]]
        users_i = set(songs_i_data[self.user_id].unique())
        for j in range(0,len(user_songs)):
            users_j = user_songs_users[j]
            users_intersection = users_i.intersection(users_j)
            if len(users_intersection) != 0:
                users_union = users_i.union(users_j)
                cooccurence_matrix[j,i] = float(len(users_intersection))/float(len(users_union))
            else:
                cooccurence_matrix[j,i] = 0
    return cooccurence_matrix
```

	item1	item2	item3	item4	item5	item6
user1						
user2						
user3						
user4						

Figure 7: Matrix M

This method does not include any personalization. Moreover, majority of songs have too few listeners so they are least likely to be recommended. We have not used play count information in final result as they did not give good result because similarity model is biased to a few songs played multiple times and calculation noise was generated by a few very popular songs.

SVD MODEL

Listening histories are influenced by a set of factors specific to the domain (e.g. genre, artist). These factors are in general not at all obvious and we need to infer those so-called latent factors from the data. Users and songs are characterized by latent factors.

Hence, to handle such a large amount of data, we build a sparse matrix from user-song triplets and directly operate on the matrix, instead of looping over millions of songs and users. We used truncated SVD for dimensionality reduction.

We used SVD algorithm in this model as follows:

Firstly, we decompose Matrix M into latent feature space that relates user to songs.

```
def compute_svd(urm, K):
    U, s, Vt = svds(urm, K)

    dim = (len(s), len(s))
    S = np.zeros(dim, dtype=np.float32)
    for i in range(0, len(s)):
        S[i,i] = mt.sqrt(s[i])

    U = csc_matrix(U, dtype=np.float32)
    S = csc_matrix(S, dtype=np.float32)
    Vt = csc_matrix(Vt, dtype=np.float32)

    return U, S, Vt

def compute_estimated_matrix(urm, U, S, Vt, uTest, K, test):
    rightTerm = S*Vt
    max_recommendation = 250
    estimatedRatings = np.zeros(shape=(MAX_UID, MAX_PID), dtype=np.float16)
    recomendRatings = np.zeros(shape=(MAX_UID,max_recommendation ), dtype=np.float16)
    for userTest in uTest:
        prod = U[userTest, :]*rightTerm
        estimatedRatings[userTest, :] = prod.todense()
        recomendRatings[userTest, :] = (-estimatedRatings[userTest, :]).argsort()[:max_recommendation]
    return recomendRatings

def show_recomendations(uTest, num_recomendations = 10):
    for user in uTest:
        print('-'*70)
        print("Recommendation for user id {}".format(user))
        rank_value = 1
        i = 0
        while (rank_value <  num_recomendations + 1):
            so = uTest_recommended_items[user,i:i+1][0]
            if (small_set.user[(small_set.so_index_value == so) & (small_set.us_index_value == user)].count()==0):
                song_details = small_set[(small_set.so_index_value == so)].\
                    drop_duplicates('so_index_value')[['title','artist_name']]
                print("The number {} recommended song is {} BY {}".format(rank_value,
                                                                          list(song_details['title'])[0],
                                                                          list(song_details['artist_name'])[0]))

                rank_value+=1
            i += 1
```

Though the theory behind SVD is quite compelling, this is not enough data for the algorithm to arrive at a good prediction. The median number of songs in a users play count history is fourteen to fifteen; this sparseness does not allow the SVD objective function to converge to a global optimum.

6. TESTING

The trained model is tested with 10000 songs to get accurate predictions.

```
K = 50
urm = data_sparse
MAX_PID = urm.shape[1]
MAX_UID = urm.shape[0]

U, S, Vt = compute_svd(urm, K)
uTest = [4,5,6,7,8,873,23]

uTest_recommended_items = compute_estimated_matrix(urm, U, S, Vt, uTest, K, True)

show_recomendations(uTest)
```

7. RESULTS

The proposed system takes in song id and predicts the related songs recommendations to the user. Collaborative filtering algorithm is observed to give best results. The SVD based latent factor model gives better results than popularity-based model. The popularity-based model is not able to give personalised recommendations. It lags behind collaborative filtering algorithm because the matrix was too sparse which prevented objective functions to converge to global optimum.

```
uTest = [1724]

print("Predictied ratings:")
uTest_recommended_items = compute_estimated_matrix(urm, U, S, Vt, uTest, K, True)
show_recomendations(uTest)
```

```
Predictied ratings:
------------------------------------------------------------------------
Recommendation for user id 1724
The number 1 recommended song is OMG BY Usher featuring will.i.am
The number 2 recommended song is Crossfire BY Rick Cua
The number 3 recommended song is Cosmic Love BY Florence + The Machine
The number 4 recommended song is You've Got The Love BY Florence + The Machine
The number 5 recommended song is Drop The World BY Lil Wayne / Eminem
The number 6 recommended song is Day 'N' Nite BY Kid Cudi Vs Crookers
The number 7 recommended song is Secrets BY OneRepublic
The number 8 recommended song is Victoria (LP Version) BY Old 97's
The number 9 recommended song is Use Somebody BY Kings Of Leon
The number 10 recommended song is Sample Track 2 BY Simon Harris
```

Figure 8 :Result

8. CONCLUSION

Music services like Spotify need an efficient way to manage songs and help their customers to discover music by giving quality recommendations. The proposed system was developed to provide these quality recommendations to the user. This system provides various recommendations to the users based on their preferences, popularity of the song and genre of the song.

There are many approaches to this problem, but collaborative filtering is observed to give more quality recommendations. This system has been initially trained and tested on 10000 songs, if required it can be further extended to more 100s of songs of various genres.

9. FUTURE WORK

• Run the algorithms on a distributed system, like Hadoop or Condor, to parallelize the computation, decrease the runtime and leverage distributed memory to run the complete MSD.

•Combine different methods and learn the weightage for each method according to the dataset

• Automatically generate relevant features

• Develop more recommendation algorithms based on different data (e.g. the how the user is feeling, social recommendation, etc)

10. REFERENCES

[1] The million song dataset (http://millionsongdataset.com/)

[2] T. Bertin et al., The Million Song Dataset, Proc. Of the 12th International Society for Music Information Retrieval Conference, 2011.