**Music Recommendation System**

**INT 404 ARTIFICIAL INTELLIGENCE**

**Assignment 2019-2020**

**Lovely Professional University**

Group no. 19

Name : Uttam veeturi Name : Md. Barkatullah Name : Deepak K. Chaurasia

Reg no. : 11802235 Reg no. :11802341 Reg no. : 11802530

Roll no. 55 Roll no. : 56 Roll no. : 57

**Description:**

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 music recommendation.

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 simpleandintuitive. Collaborativeﬁlteringalgorithmswhich predict (ﬁltering) taste of a user by collecting preferences and tastes from many other users (collaborating) is also implemented.

**Data**

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, audio 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(280GB)of millions of songs[7]. We decided that information like year, duration, dance-ability, etc. may distinguish a song most from other songs.

ALGORITHMS :

There are 3 types of recommendation system: content-based, collaborative and popularity.

1. **Popularity Based** - 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.

The code for the Recommender Systems model is below. This system is a naive approach and not personalized. It first get a unique count of user\_id (ie the number of time that song was listened to in general by all user) for each song and tag it as a recommendation score. The recommend function then accept a user\_id and output the top ten recommended song for any given user. Keeping in my that since this is the naive approach, the recommendation is not personalized and will be the same for all users.

#Class for Popularity based Recommender System modelclass popularity\_recommender\_py():

def \_\_init\_\_(self):

self.train\_data = None

self.user\_id = None

self.item\_id = None

self.popularity\_recommendations = None

#Create the popularity based recommender system model

def create(self, train\_data, user\_id, item\_id):

self.train\_data = train\_data

self.user\_id = user\_id

self.item\_id = item\_id

#Get a count of user\_ids for each unique song as recommendation score

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)

#Sort the songs based upon recommendation score

train\_data\_sort = train\_data\_grouped.sort\_values(['score', self.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 10 recommendations

self.popularity\_recommendations = train\_data\_sort.head(10)

#Use the popularity based recommender system model to

#make recommendations

def recommend(self, user\_id):

user\_recommendations = self.popularity\_recommendations

#Add user\_id column for which the recommendations are being generated

user\_recommendations['user\_id'] = user\_id

#Bring user\_id column to the front

cols = user\_recommendations.columns.tolist()

cols = cols[-1:] + cols[:-1]

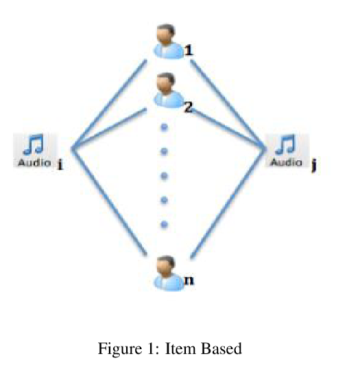
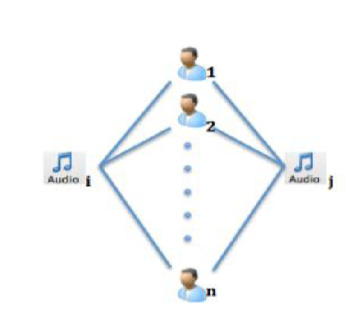
user\_recommendations = user\_recommendations[cols]

return user\_recommendations

1. **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 itembased models.

* Item based models:

 user based

Item-item filtering approach involves defining a co-occurrence matrix based on a song a user likes. We are seeking to answer a question, for each song, what a number of time a user, who have listened to that song, will also listen to another set of other songs. To further simplify this, based on what you like in the past, what other similar song that you will like based on what other similar user have liked.

Notice that inside the recommender system’s source code, the **generate\_top\_recommendations function** calculated a weighted average of the scores in cooccurence matrix for all user song. This cooccurence matrix will tend to be sparse matrix because it’s not possible to predict if a user like a particular song, whether or not he/she will like a million other song. The possibility is so vast. Using our model, we will be able to predict the list of song that a user will like.

Code :

#Print the songs for the user in training data

user\_id = users[5]

user\_items = is\_model.get\_user\_items(user\_id)

#

print("------------------------------------------------------------------------------------")

print("Training data songs for the user userid: %s:" % user\_id)

print("------------------------------------------------------------------------------------")

for user\_item in user\_items:

print(user\_item)

print("----------------------------------------------------------------------")

print("Recommendation process going on:")

print("----------------------------------------------------------------------")

#Recommend songs for the user using personalized model

is\_model.recommend(user\_id)

Output :

Training data songs for the user userid: 4bd88bfb25263a75bbdd467e74018f4ae570e5df:

------------------------------------------------------------------------------------

Just Lose It - Eminem

Without Me - Eminem

16 Candles - The Crests

Speechless - Lady GaGa

Push It - Salt-N-Pepa

Ghosts 'n' Stuff (Original Instrumental Mix) - Deadmau5

Say My Name - Destiny's Child

My Dad's Gone Crazy - Eminem / Hailie Jade

The Real Slim Shady - Eminem

Somebody To Love - Justin Bieber

Forgive Me - Leona Lewis

Missing You - John Waite

Ya Nada Queda - Kudai

----------------------------------------------------------------------

Recommendation process going on:

----------------------------------------------------------------------

No. of unique songs for the user: 13

no. of unique songs in the training set: 4483

Non zero values in cooccurence\_matrix :2097

1. **Matrix Factorization based Recommender System.**

This type of recommender system uses what is called a Singular Value Decomposition (SVD) factorized matrix of the original similarity matrix to build recommender system.To compute SVD and recommendations, we use the following code:

**Code:**

#constants defining the dimensions of our User Rating Matrix (URM) MAX\_PID = 4

MAX\_UID = 5

#Compute SVD of the user ratings matrix

def computeSVD(urm, K):

U, s, Vt = sparsesvd(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(np.transpose(U), dtype=np.float32)

S = csc\_matrix(S, dtype=np.float32)

Vt = csc\_matrix(Vt, dtype=np.float32)

return U, S, Vt

In this code, U represents user vector, S represents the item vector.Vt represent the joint of these two vectors as collection of points (ie vector) in 2 dimensional spaces. We’re going to use these vectors to measure the distance from one user’s preferences to another user’s preferences.

In another word, we are vectorizing matrices in order to compute the distance between matrices.

Assume we have a user song matrix below:

Song0 Song1 Song2 Song3

User0 3 1 2 3

User1 4 3 4 3

User2 3 2 1 5

User3 1 6 5 2

User4 0 0 5 0

Once we perform SVD, the output is going to be vectors and measuring distance between vectors gives us recommendation.

Code for SVD :

#Compute estimated rating for the test user

def computeEstimatedRatings(urm, U, S, Vt, uTest, K, test):

rightTerm = S\*Vt

estimatedRatings = np.zeros(shape=(MAX\_UID, MAX\_PID), dtype=np.float16)

for userTest in uTest:

prod = U[userTest, :]\*rightTerm

#we convert the vector to dense format in order to get the #indices

#of the movies with the best estimated ratings

estimatedRatings[userTest, :] = prod.todense()

recom = (-estimatedRatings[userTest, :]).argsort()[:250]

return recom

#Used in SVD calculation (number of latent factors)

K=2

#Initialize a sample user rating matrix

urm = np.array([[3, 1, 2, 3],[4, 3, 4, 3],[3, 2, 1, 5], [1, 6, 5, 2], [5, 0,0 , 0]])

urm = csc\_matrix(urm, dtype=np.float32)

#Compute SVD of the input user ratings matrix

U, S, Vt = computeSVD(urm, K)

#Test user set as user\_id 4 with ratings [0, 0, 5, 0]

uTest = [4]

print("User id for whom recommendations are needed: %d" % uTest[0])

#Get estimated rating for test user

print("Predictied ratings:")

uTest\_recommended\_items = computeEstimatedRatings(urm, U, S, Vt, uTest, K, True)

print(uTest\_recommended\_items)

output :

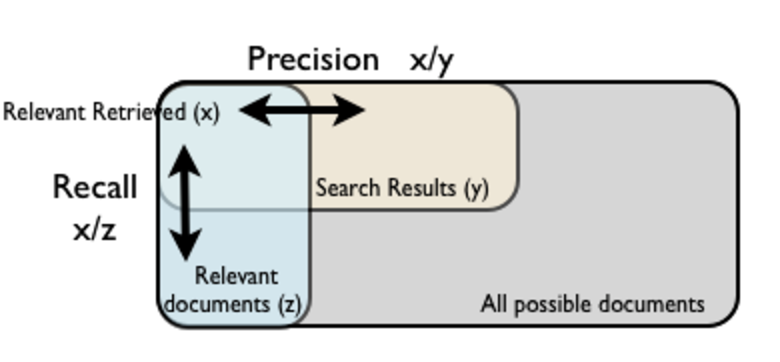
User id for whom recommendations are needed: 4

Predictied ratings:

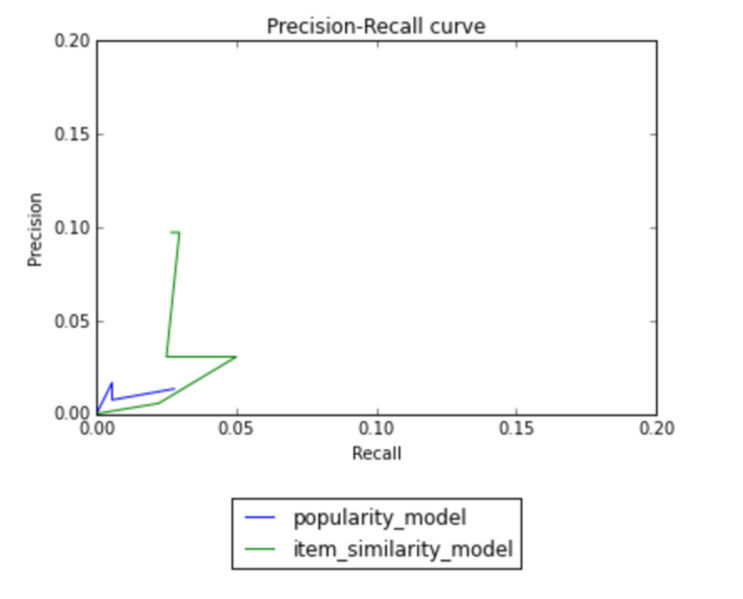
[0 3 2 1]

**Result and Conclusion:**

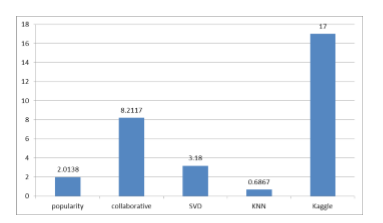
To quantitatively measure the performance of recommender system, we use three different metrics: Precision , Recall and F-1 Score:



Precision-recall curve :



Also,

****

We got best results for memory based collaborative ﬁltering algorithm. Our SVD based latent factor model gives better results than popularity based model. It lags behind collaborative ﬁltering algorithm because the matrix was too sparse which prevented objective functions to converge to global optimum.

**SOURCES :**

1/Siraj Raval’s Deep Learning Foundation Nanodegree (<https://www.udacity.com/course/deep-learning-nanodegree-foundation--nd101>)

2/<https://www.youtube.com/watch?v=18adykNGhHU>

3/Implementing your own recommender systems in Python(http://online-dev.cambridgecoding.com/notebooks/eWReNYcAfB/implementing-your-own-recommender-systems-in-python-2)