**Song recommender system**

**INTROUCTION**

During everyday life people always need to take advice from people which movie to watch on upcoming holidays or is there any interesting new products available on amazon. Under this context, we are building a song recommender system. That will recommend songs to the users based on their interests after doing researches on their listening histories.

**PREVIOUS RELATED WORKS AND STUDIES**

* Recommendation system is one of the most discussed topic in the previous years, whether it’s about recommendation of products on amazon, youtube videos suggestion or Spotify song recommendation.
* One of the remarkable development in the field of recommendation is the NETFLIX challenged competitors to come up with more efficient movie recommender system based on the user’s watching history,
* The winning team used a combination of various methods, but one of the most successful algorithm was Latent Factor Model and Matrix Factorization Model.
* We will try to implement this using Collaborative filtering method , which uses user-based and item-based similarity for recommendation.
* Another set of methods which show promises now a days is Matrix Factorization method for recommendations.

**ML TOOL USED**

* Collaborative Filtering.
* Co-ocurrence Matrix.
* Jaccard Index.
* K-Nearest Neighbor.
* User and Item similarity based.
* Tkinter python library for GUI.
* Numpy, Pandas, Matplotlib and Scikit-learn libray.

**MODEL DIAGRAM/DFD**

A Data Flow Diagram file(**DFD.doc**) is attached seperately.

**ALGORITHM/FLOW CHART**

Collaborative Filtering:-

**Algorithm Co-Ocurrence Matrix**:-

*Get users for all songs in user\_songs.  
  
Initialize the item cooccurence matrix of size   
 len(user\_songs) X len(songs)  
  
Calculate similarity between user songs and all unique songs  
in the training data*

**For each song in all songs, do-**  
 *Calculate unique listeners (users) of song (item) i* **for** each item in user’s playing history:  
   
 *#Get unique listeners (users) of song (item)*    
 *#Calculate intersection of listeners of songs item and song*   
 *#Calculate cooccurence\_matrix[i,j] as Jaccard Index* **if** intersection not equals zero Do  
 *#Calculate union of listeners of songs i and j*   
 cooccurence\_matrix[j,i] = intersection(users)/union(users)  
 **else**  
 cooccurence\_matrix[j,i] = 0  
   
  
**return** cooccurence\_matrix

**Item simality based recommendation:-**

Item\_based(song\_id)\_

For song\_id

Calculate co-occurrence Matrix

Sort all songs in descending order based on the value in the matrix.

Return Top 10 songs

**User similarity based recommendation:-**

User\_similarity(user\_id):

user\_songs =user\_id’s listening history

Construct co-ocurrence Matrix(user\_songs)

Take normalised weighted average of all song columns

Sort all songs in descending order based on the value in the matrix.

Return Top 10 songs.

**DATASET and REFERENCE**

**Song\_triplet(10000.txt)**

|  |  |  |
| --- | --- | --- |
| User\_id | Song\_id | Listen\_count |

**Song\_info(song\_data.csv)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Song\_id | Title | Release | Artist\_name | Years |

**References:-**

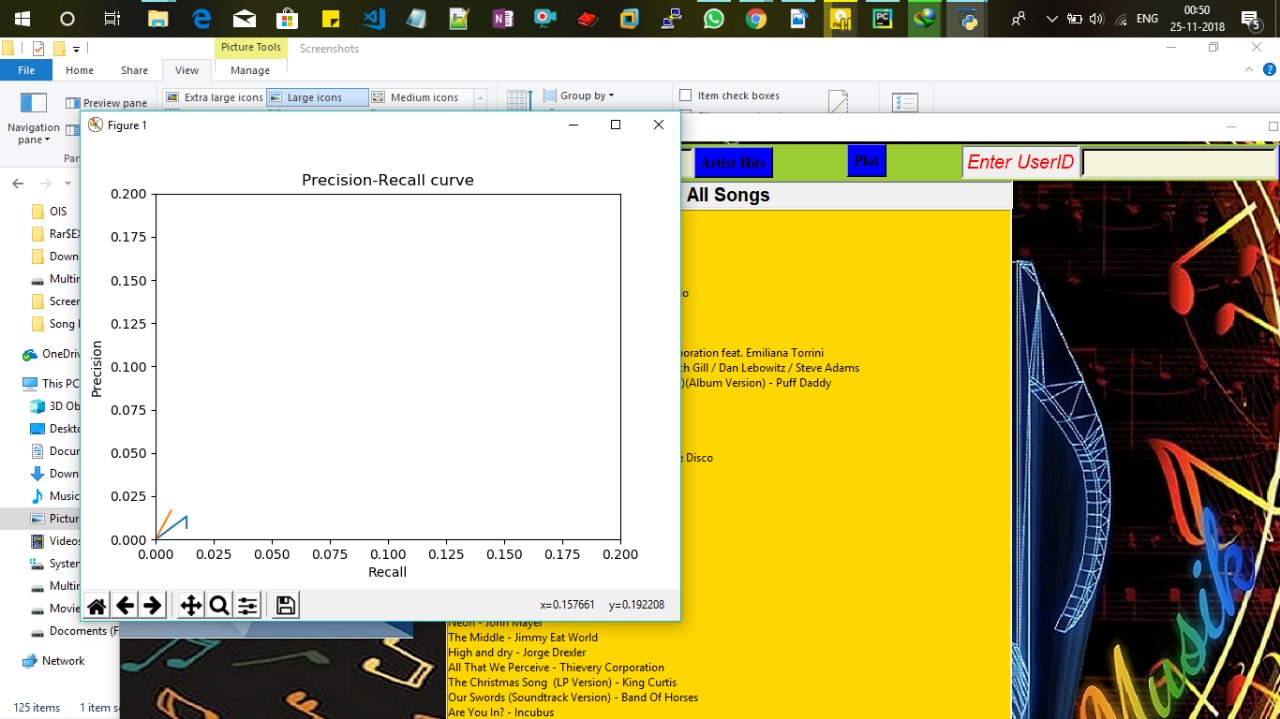
|  |  |
| --- | --- |
| 10000.TXT | ‘https://static.turi.com/datasets/millionsong/10000.txt’ |
| SONG\_DATA.CSV | 'https://static.turi.com/datasets/millionsong/song\_data.csv' |

**SOURCE CODE**

|  |
| --- |
| **import** pandas **from** sklearn.model\_selection **import** train\_test\_split **import** numpy **as** np **import** time **from** sklearn.externals **import** joblib **import** Recommenders **as** Recommenders **import** Evaluation **as** Evaluation  *#This step might take time to download data from external sources* triplets\_file = **'10000.txt'** songs\_metadata\_file = **'song\_data.csv'** song\_df\_1 = pandas.read\_table(triplets\_file,header=**None**) song\_df\_1.columns = [**'user\_id'**, **'song\_id'**, **'listen\_count'**]  *#Read song metadata* song\_df\_2 = pandas.read\_csv(songs\_metadata\_file)  *#Merge the two dataframes above to create input dataframe for recommender systems* song\_df = pandas.merge(song\_df\_1, song\_df\_2.drop\_duplicates([**'song\_id'**]), on=**"song\_id"**, how=**"left"**)  *#CREATE SUBSETOF DAATASET* song\_df=song\_df.head(10000) *#Merge song title and artist\_name columns to make a merged column* song\_df[**'song'**] = song\_df[**'title'**].map(str) + **" - "** + song\_df[**'artist\_name'**]  *#CREATE A SONG RECOMMENDER* train\_data,test\_data=train\_test\_split(song\_df,test\_size=0.20,random\_state=0) *#print(train\_data.head(5))  #============================popularity based recommendation============================* pm = Recommenders.popularity\_recommender\_py() pm.create(train\_data, **'user\_id'**, **'song'**)   *#============================item\_similarity based recommendation=================* is\_model = Recommenders.item\_similarity\_recommender\_py() is\_model.create(train\_data, **'user\_id'**, **'song'**) *#==============================plot graph============================= #Define what percentage of users to use for precision recall calculation* user\_sample = 0.05  *#Instantiate the precision\_recall\_calculator class <pm-popularity based recc. model>,<is\_model-Item similarity based>* pr = Evaluation.precision\_recall\_calculator(test\_data, train\_data, pm, is\_model)  *#Call method to calculate precision and recall values* (pm\_avg\_precision\_list, pm\_avg\_recall\_list, ism\_avg\_precision\_list, ism\_avg\_recall\_list) = pr.calculate\_measures(user\_sample)   *#Code to plot precision recall curve* **import** pylab **as** pl  *#Method to generate precision and recall curve* **def** plot\_precision\_recall(m1\_precision\_list, m1\_recall\_list, m1\_label, m2\_precision\_list, m2\_recall\_list, m2\_label):  pl.clf()  pl.plot(m1\_recall\_list, m1\_precision\_list, label=m1\_label)  pl.plot(m2\_recall\_list, m2\_precision\_list, label=m2\_label)  pl.xlabel(**'Recall'**)  pl.ylabel(**'Precision'**)  pl.ylim([0.0, 0.20])  pl.xlim([0.0, 0.20])  pl.title(**'Precision-Recall curve'**)  *#pl.legend(loc="upper right")* pl.legend(loc=9, bbox\_to\_anchor=(0.5, -0.2))  pl.show()  *#xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx* **from** tkinter **import** \* **from** PIL **import** ImageTk,Image panel = Tk() panel.title(**"musik"**)  *#===============================IMAGES==================================* img=ImageTk.PhotoImage(Image.open(**'pic1.jpg'**)) root=Label(panel,image=img,width=1600,height=800) root.pack(side=TOP,fill=BOTH,expand=YES)    frame = Frame(root) frame.pack()  Id= StringVar() lis=StringVar() art=StringVar() **def** plot\_data():  plot\_precision\_recall(pm\_avg\_precision\_list, pm\_avg\_recall\_list, **"popularity\_model"**,ism\_avg\_precision\_list, ism\_avg\_recall\_list, **"item\_similarity\_model"**)  **def** search\_ar():  sgsl = art.get()  root3 = Tk()  print(sgsl)  root3.title(**"Musik"**)  L1 = Label(root3, text=**"Artist greatest hits"**, font=(**" Comic Sans MS"**, 15, **"bold"**), relief=**"sunken"**)  L1.pack()   *# Show output of similer song list* Lst4 = Listbox(root3, width=60, bg=**'gold'**, selectforeground=**'cyan'**)  Lst4.pack(side=RIGHT, expand=0.5, fill=Y)   arts=song\_df[song\_df[**'artist\_name'**]== sgsl]  *#POPULAR ARTIST* song\_artist = arts.groupby([**'song'**]).agg({**'listen\_count'**: **'count'**}).reset\_index()  similar=song\_artist.sort\_values([**'listen\_count'**],ascending =**False**)  *#similar=similar['song']+" - " + similar['listen\_count'].map(str)* **for** q **in** similar[**'song'**]:  Lst4.insert(END, q)  root3.configure(bg=**'skyblue'**)  root3.mainloop()    **def** search\_s():  sgsl = lis.get()  root3 = Tk()  root3.title(**"Musik"**)  L1 = Label(root3, text=**"Similar Musics are here"**, font=(**" Comic Sans MS"**, 15, **"bold"**), relief=**"sunken"**)  L1.pack()   *# Show output of similer song list* Lst4 = Listbox(root3, width=60, bg=**'gold'**, selectforeground=**'cyan'**)  Lst4.pack(side=RIGHT, expand=0.5, fill=Y)  sim=is\_model.get\_similar\_items(sgsl)  sim=sim[**'rank'**].map(str)+**" - "** + sim[**'song'**]  Lst4.insert(1,**' '**+sgsl)  ct=2  **for** x **in** sim:  Lst4.insert(ct,x)  ct+=1   root3.configure(bg=**'skyblue'**)  root3.mainloop()   **def** person():    entry=Id.get()   *#Got The User Id in "entry" Perform ML   #Show output From here  #print(entry)* root2 = Tk()  root2.title(**"musik"**)   b\_frame = Frame(root2,bg=**"skyblue"**)  b\_frame.pack()   *#show User Id At Top* L1 = Label(b\_frame, text=**" Result For User:-<<"**+entry+**">>"**, font=(**" Comic Sans MS"**, 15, **"bold"**, **"underline"**), relief=**"sunken"**)  L1.pack()   L2 = Label(b\_frame, text=**"SONG playing History"**, font=(**" Comic Sans MS"**, 15, **"bold"**), relief=**"sunken"**,width=40)  L2.pack(side=LEFT,expand=1)   L3 = Label(b\_frame, text=**"Only for you"**, font=(**" Comic Sans MS"**, 15, **"bold"**), relief=**"sunken"**,width=40)  L3.pack(side=RIGHT,expand=1)   L4 = Label(b\_frame, text=**"Popular Songs"**, font=(**" Comic Sans MS"**, 15, **"bold"**), relief=**"sunken"**, width=40)  L4.pack(side=RIGHT, expand=1)   *#Output For perticuler User  #There'll be 3 lists under "root2"  #"Only For You","Populer Songs","played Songs"  #enter items for PLAYING HISTORY* Lst = Listbox(root2,width=60,bg=**'gold'**,selectforeground=**'cyan'**)  Lst.pack(side=LEFT, fill=Y,expand=0.5)  user\_items = is\_model.get\_user\_items(entry)  cnt=1  **for** x **in** user\_items:  Lst.insert(cnt,x)  cnt+=1    *#enter items for ONLY FOR YOU* Lst1 = Listbox(root2,width=60,bg=**'gold'**,selectforeground=**'cyan'**)  Lst1.pack(side=RIGHT,expand=0.5, fill=Y)  out=is\_model.recommend(entry)  out=out[**'rank'**].map(str)+**" - "**+out[**'score'**].map(str) + **" - "** + out[**'song'**]  cnt=1  **for** x **in** out:  Lst1.insert(cnt,x)  cnt+=1   *#enter items for POPULAR SONGS* Lst2 = Listbox(root2,width=60,bg=**'gold'**,selectforeground=**'cyan'**)  Lst2.pack(side=LEFT,fill=Y,expand=0.5)  output=pm.recommend(entry)  output=output[**'Rank'**].map(str)+**" - "**+output[**'score'**].map(str)+**" - "**+output[**'song'**]  count=1  **for** x **in** output:  Lst2.insert(count,x)  count+=1      root2.geometry(**"1300x660+120+120"**)  root2.configure(bg=**'skyblue'**)  root2.mainloop()    Toolbar=Frame(root,bg=**"yellow green"**,relief=**"sunken"**)  L1 = Label(Toolbar, text=**"Search Songs"**, font=(**" Comic Sans MS"**, 15, **"italic"**),fg=**'red'**, relief=**"sunken"**) L1.pack(side=LEFT)  entry\_2 = Entry(Toolbar, textvariable= lis, bd=4, font=(**"Times"**, 15, **"bold"**), bg=**'beige'**, fg=**'Brown'**) entry\_2.pack(side=LEFT)  button1 = Button(Toolbar, text=**"Search"**, font=(**"Times"**, 12, **"bold"**), bg=**'Blue'**,command=search\_s) button1.pack(side=LEFT)  entry\_2 = Entry(Toolbar, textvariable=art, bd=4, font=(**"Times"**, 15, **"bold"**), bg=**'beige'**, fg=**'Brown'**) entry\_2.pack(side=LEFT)  button3 = Button(Toolbar, text=**"Artist Hits"**, font=(**"Times"**, 12, **"bold"**), bg=**'Blue'**,command=search\_ar) button3.pack(side=LEFT)  button = Button(Toolbar, text=**"Dig In"**, font=(**"Times"**, 12, **"bold"**), bg= **'Blue'** , command= person,padx=4,pady=4 ) button.pack(side=RIGHT)  entry\_1=Entry(Toolbar,textvariable= Id, bd=5, font= (**"Times"**, 15, **"bold"**),bg=**'beige'**,fg=**'Brown'**) entry\_1.pack(side=RIGHT)  L3 = Label(Toolbar, text=**"Enter UserID"**, font=(**" Comic Sans MS"**, 15, **"italic"**),fg=**'red'**, relief=**"sunken"**,padx=4,pady=4) L3.pack(side=RIGHT)  button2 = Button(Toolbar, text=**"Plot"**, font=(**"Times"**, 12, **"bold"**), bg=**'Blue'**,padx=2,pady=2,command=plot\_data) button2.pack()  Toolbar.pack(side=TOP,fill=X)  frame = Frame(root) frame.pack()  Lb2 = Label(frame, text=**"All Songs"**, font=(**" Comic Sans MS"**, 15, **"bold"**), relief=**"sunken"**, width=50) Lb2.pack(expand=1)   *#output of First Page #All Songs and Hit Songs* List1= Listbox(root,width=100,bg=**'gold'**,selectforeground=**'cyan'**) songlist=song\_df[**'song'**] **for** q **in** songlist:  List1.insert(END,q)  List1.pack(side= LEFT,expand=0.5,fill=Y)   panel.geometry(**"1300x660+120+120"**) panel.configure(bg=**'skyblue'**) frame.configure(bg=**'#26261f'**) panel.mainloop() |

**RESULT & ACCURACY(PLOT)**

We have successfully build a song recommender system model which gives personalized song based on user’s listening history. We recommended Popular songs to the user ,Artist top hits , Recommended set of similar songs based on user’s playing history(PERSONALISED RECOMMENDATION). At the end we have plotted a graph showing popularity based and item based song recommendation.



We can see clearly in the plot above that item based recommendation gives more personalized recommendation as it is solely based on listening history compared to popularity based recommendation.

Since our dataset consists of playing history of different user’s and not the set of recommendation for a user so we are unable to calculate precision in words, instead we have plotted graph for the same.

**Application of proposed work:**

The song recommender system which we have build consists of following features:-

* Trending Song List (top hits).
* Artist’s best song listing.
* Personalized Recommendation based on an user’s song listening history.
* Search similar songs.

**Advantage and dis-advantage showing effectiveness and limitations**

**ADVANTAGE:**

* We have successfully recommended similar songs to the user by selecting similar group of user’s having similar interest. (user based similarity).
* We also provided trending songs, artist’s top hits based on listen\_count of various songs by different users.
* You can also search similar song from the library. We have implemented this by selecting all those songs having similar group of listeners.

i.e Item based similarity.

* Last but not least an amazing GUI containing all these features has also been provided.

**DISVANTAGE:-**

* Due to absence of genre in our dataset we are unable to implement algorithm for genre based recommendations.
* Since there is no limit of perfection, obviously accuracy of recommendation can be enhanced further using better algorithms(Matrix Factorization, Latent Based model etc.)

**Future scope**

Data is increasing day by day, If we will have a dataset containing lots of attributes(genre, time of playing , place at which song is played, user’s mood ….). Using all these attributes we will be able to recommend most relevant songs to the user’s based on their mood, daily routine, time of the day(morning masala ..). Thus there is a lot of work to be done in the future to enhance recommendation accuracy.

**Conclusion**

We have tried to build a song recommender model which recommend song to user’s based on their playing history. Our model recommend both popularity and personalized songs .However, accuracy can be improved further by implementing better algorithms. Furthermore, we believe that

running our models on larger matrices would have

yielded a significant improvement in performance.