# General overview

The objective as noted earlier was to develop a deep learning model and evaluate its performance against memory-based models i.e., cosine similarity and non-negative matrix factorization. Before developing the models, we collected the data from the respective websites as indicated in the analysis book. The following subsections provide an overview of the code sections and their respective functions.

Note: Each of the steps that is, Exploratory data analysis, data preparation, and analysis steps are replicated for each of the three data frames. To avoid cluttering, we will provide an explanation of what the codes do. Also, where, the computation is not obvious, we have included comments showing what the next item does.

The analysis will be done on about two main levels including exploratory for us to understand what the data is about and modelling for the development of the proposed models (see below).

# Models and Evaluation Metrics

We propose three models:

1. Implicit-based recommender systems where we implement a simple cosine similarity model and a non-negative matrix factorization model
2. Deep learning-based neural collaborative filtering

The performance of the models is evaluated using the mean percentile score on the test set as well as the recall of the models.

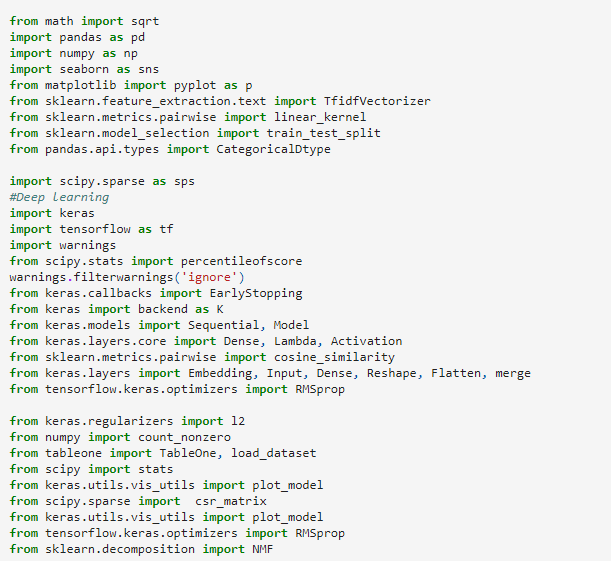
|  |  |  |
| --- | --- | --- |
| Evaluation Metric | Summary | Formula |
| Recall | Computes the rate at which the recommender makes positive recommendations |  |
| Mean Percentile Ranking | Computes the  percentile of an item *i* provided in the list of all the items, and then calculates the average percentile. Primarily, this implies that if one  made an allocation of recommendations at random, then the expected value of the MPR would be 50%. | Where:  r(i,j) denotes the rating of item *i* by user *u*  ui denotes the percentile rank of *i* in an ordered list such that ui = 0% implies that *i* is at the top of the list |

The steps involved for the implicit-based cosine similarity and deep learning models include:

# Code Explanation per section

Since the codes were generally replicated for the three datasets, we will explain the functionality for the codes used for the 100K datasets to provide an understanding of the whole codes.

##### Import packages



##### Importing data

We imported data using pandas read\_csv function as shown below.

*# Reading scores file*

scores **=** pd**.**read\_csv('ratings.csv', sep**=**',', encoding**=**'latin-1', usecols**=**['userId','movieId','rating','timestamp'])

*# Reading movies file*

movies **=** pd**.**read\_csv('movies.csv', sep**=**',', encoding**=**'latin-1', usecols**=**['movieId','title','genres'])

##### Examine the information provided by the data including the number of user-item interaction

ct **=** 'category' *#Define global category descriptor*

usrs **=** np**.**sort(scores**.**userId**.**unique())**.**tolist() *# Get the unique customers*

mvs **=** list(scores**.**movieId**.**unique()) *# Get the unique products that were watched*

rtng **=** list(scores**.**rating) *# All of the watch history*

r **=** scores**.**userId**.**astype(ct, usrs)**.**cat**.**codes

*# pull row indices*

cols **=** scores**.**movieId**.**astype(ct, mvs)**.**cat**.**codes

*# pull column indices*

usr\_itm **=** sps**.**csr\_matrix((rtng, (r, cols)), shape**=**(len(usrs), len(mvs)))

mtrx\_sz **=** usr\_itm**.**shape[0]**\***usr\_itm**.**shape[1] *# Total possible links between users and movies*

times\_watched **=** len(usr\_itm**.**nonzero()[0]) *# Number of movievs interacted with*

*#sparsity*

sprs **=** 100**\***(1 **-** (1.0**\***times\_watched**/**mtrx\_sz))

##### Exploratory Data Analysis(EDA)

This section included examining the various demographical aspects of the data as shown below.

##### Most popular genres of movie released

p**.**figure(figsize**=**(20,7))

#Plot the number of times movies related to a given genre were watched

#Split the movies by genre, some movies were related to multiple genres

gnr\_lst **=** dfmvs1['genres']**.**apply(**lambda** mv\_lst : str(mv\_lst)**.**split("|"))

gnr\_cnt **=** {}

**for** mv\_lst **in** gnr\_lst:

**for** genre **in** mv\_lst:

**if**(gnr\_cnt**.**get(genre,**False**)):

gnr\_cnt[genre]**=**gnr\_cnt[genre]**+**1

**else**:

gnr\_cnt[genre] **=** 1

#Remove the movies that had no genre

gnr\_cnt**.**pop('(no genres listed)')

#Sort the disctionary having the number of times a given genre occurred

gnr\_cnt **=** {k: v **for** k, v **in** sorted(gnr\_cnt**.**items(), key**=lambda** item: item[1], reverse**=True**)}

#use a bar chart

p**.**bar(gnr\_cnt**.**keys(),gnr\_cnt**.**values(),color**=**'steelblue')

p**.**title('Movie by genre')

#rotate the xticks

p**.**xticks(rotation **=** 45)

#Add the label for x axis

p**.**xlabel('Genre')

p**.**ylabel('Number of movies')

#Adds a grid to the plot

p**.**grid()

p**.**show()

##### Distribution of users rating

p**.**figure(figsize**=**(12,7))

*#plot distribution of rating with the kde chart in sns*

sns**.**kdeplot(df\_rtngs["rating"], color **=** 'steelblue');

p**.**title('Distribution of movie scores')

p**.**grid()

p**.**show()

##### Examine shape of the datasets

print("Shape of the dataframes: \n"**+** " Rating DataFrame: "**+** str(df\_rtngs**.**shape)**+**"\n Movies DataFrame"**+** str(dfmv**.**shape))

##### Merge datasets

*#merge the movie and ratings dataset to get ratings of the respective movies*

combine\_mv\_scs **=** pd**.**merge(dfmv, df\_rtngs, on**=**'movieId', how**=**'inner')

*#drop the timestamp column to remain with relevant attrs*

combine\_mv\_scs **=** combine\_mv\_scs**.**drop('timestamp', axis**=**1)

#Grouping the rating based on user

*#Group by the userid*

scores\_grouped\_by\_usrs **=** combine\_mv\_scs**.**groupby('userId')**.**agg([np**.**size, np**.**mean])

*#Remove movieid*

scores\_grouped\_by\_usrs **=** scores\_grouped\_by\_usrs**.**drop('movieId', axis **=** 1)

#Top 10 users who have rated most of the movies

*#plot the top 10 movies*

scores\_grouped\_by\_usrs['rating']['size']**.**sort\_values(ascending**=False**)**.**head(10)**.**plot(kind **=** 'bar', figsize **=** (11,6))

p**.**title('Top 10 movie reviews (users with most scores)')

p**.**grid()

p**.**show()

*#Group by movie id to get the mean rating of each movie*

scores\_mvs **=** combine\_mv\_scs**.**groupby('movieId')**.**agg([np**.**mean], np**.**size)

##### Top 10 users who have rated most of the movies

Group ratings by users and select the top 10 users

ratings\_grouped\_by\_users['rating']['size']**.**sort\_values(ascending**=False**)**.**head(10)**.**plot(kind **=** 'bar', figsize **=** (10,5))

Plot using a bar plot

plt**.**title('Top 10 movie reviews (users with most ratings)')

plt**.**grid()

plt**.**show()

##### Movies with high average rating

*#Plot movies with the highest average rating*

scores\_mvs['rating']['mean']**.**sort\_values(ascending**=False**)**.**head(10)**.**plot(kind**=**'bar',

figsize**=**(7,6), color **=** 'orange');

p**.**title('Movies with the highest scores')

p**.**grid()

p**.**show()

##### Movies with low average rating

*#Get movies that are rated less than 1.5*

least\_rated\_mvs **=** scores\_mvs['rating']['mean']**<** 1.5

*#Subset the movies using the mentioned criterion*

rated\_low **=** scores\_mvs[least\_rated\_mvs]

*#Overview of 10 movies with the lowest rating*

rated\_low**.**head(10)

*#Merge the ratings and movies dt*

dfmvs **=** dfmv**.**merge(df\_rtngs)

groupby **=** ['genres']

nonnormal **=** ['movieId']

#Select movies from the most watch movie genres, this will help handle memory issues related to using the complete data but captures most of the information

dfmvs **=** dfmvs[dfmvs['genres']**.**isin(list(gnr\_cnt**.**keys())[0:5])]

##### Compute basic data statistics

#Use tableone to compute genre-rating frequency summary

TestTable **=** TableOne(dfmvs,columns **=** ['genres', 'rating'],

groupby**=**groupby, pval**=True**)

#add the summary statistics to a csv file

TestTable**.**to\_excel('summary\_100k.xlsx')

*#Geenrate statistics related to genres and ratings*

TestTable **=** TableOne(dfmvs,columns **=** ['genres', 'rating'],

groupby**=**groupby, categorical**=**['genres'], nonnormal**=**nonnormal, pval**=True**, tukey\_test**=True**)

display(TestTable)

*#Save to excel for use in the report*

TestTable**.**to\_excel('summary\_100k\_test.xlsx')

## Modelling

### Data Preparation

##### Select train and test sets

X **=** movie\_scs[['userId', 'movieId']]**.**values

y **=** movie\_scs['rating']**.**values

*#Split*

trn, tst **=** train\_test\_split(X, tst\_size**=**0.30, random\_state**=**420)

#check shape of the data

print(trn**.**shape)

print(tst**.**shape)

The following general steps were followed for the modeling process:

##### Generate an index vector for user-item interaction

*#Order the movies and users*

c\_mv **=** CategoricalDtype(sorted(movie\_scs['movieId']**.**unique()), ordered**=True**)

c\_usrr **=** CategoricalDtype(sorted(movie\_scs['userId']**.**unique()), ordered**=True**)

##### Check the number of categories in the resulting attributes

print(len(c\_mv**.**categories))

print(len(c\_usrr**.**categories))

##### Make a copy of the data

#Making a copy will mean we have an original copy so we wont have to rerun the data import incase of variable description issues

tst **=** X\_tst**.**copy()

trn **=** X\_trn**.**copy()

tstx **=** tst**.**copy()

tstx['userId'] **=** tstx['userId']**.**astype(c\_usrr)

tstx['movieId'] **=** tstx['movieId']**.**astype(c\_mv)

tst\_pred **=** tstx**.**copy()

##### Convert the user-item interactions to sparse matrix

Define a 1's and 0's binary sparse matrix of User-movies for training set

trn\_dt **=** csr\_matrix((np**.**ones(len(trn)), (trn['userId']**.**astype(ct)**.**cat**.**codes,

trn['movieId']**.**astype(ct)**.**cat**.**codes)),

shape**=**( len(c\_usrr**.**categories), len(c\_mv**.**categories)))

print(f'{100**\***trn\_dt**.**sum()**/**(trn\_dt**.**shape[0]**\***trn\_dt**.**shape[1])}% Sparsity')

##### Train the model on the training set

The models proposed include:

*Cosine Similarity (CS)*

*Non-negative matrix factorization (NMF)*

*Neural Collaborative Filtering (NCF)*

Evaluation metrics to evaluate the quality of the recommendations:

*Recall- percentage of 1s that are found in the Test set*

*MPR- Mean Percentile Rank score*

#### Cosine Similarity

**def** cos(dt, pred):

*#Cosine similarity model to generate a matrix and determine the closest recommendation depending on the similarity grid*

*# normalize to obtain % importance to user*

scaled\_df **=** dt**.**multiply(1**/**dt**.**sum(axis**=**1))**.**tocsr()

*# implement model*

*#s = similarity*

s **=** cosine\_similarity(scaled\_df**.**transpose())

s[np**.**diag\_indices(s**.**shape[0])] **=** 0

denom **=** np**.**asarray(s**.**sum(axis**=**1))**.**reshape(**-**1)

*# implement per bit to handle memory issues*

user\_index **=** pred['userId']**.**cat**.**codes**.**values

movie\_index **=** pred['movieId']**.**cat**.**codes**.**values

number\_of\_bits **=** 30

bits **=** np**.**array\_split(np**.**arange(scaled\_df**.**shape[0]), number\_of\_bits)

*#R = result*

r **=** []

prev\_mx **=** 0

**for** i,indexx **in** enumerate(bits):

print(f'Implementing bit : {i**+**1}/{number\_of\_bits}')

scr **=** (scaled\_df[indexx]**.**dot(s)) **/** denom

scr **=** (**-**scr)**.**argsort() **/** denom**.**shape[0]

sl **=** (user\_index **>=** indexx**.**min()) **&** (user\_index **<=** indexx**.**max())

bit\_scr **=** np**.**asarray(scr[user\_index[sl] **-** prev\_mx, movie\_index[sl]])**.**reshape(**-**1)

r**.**append(bit\_scr)

prev\_mx **=** indexx**.**max() **+** 1

**return** np**.**concatenate(r)

*# Evaluate performance of the model*

Recommendations **=** cos(trn\_dt, tstx)

mpr\_cosine\_sim **=** Recommendations**.**sum()**/**len(tstx)

print('MPR:', round(mpr\_cosine\_sim,5))

r\_cosine\_sim **=** (Recommendations **<** 0.5)**.**sum()**/**len(tstx)

print('Model Recall rate:',round(r\_cosine\_sim, 5))

#### Neural Collaborative Filtering

*# Create the training Set*

sample\_size\_with\_no\_links **=** int(len(trn)**\***0.2)

n\_usrs **=** c\_usrr**.**categories**.**shape[0]

num\_of\_movies **=** c\_mv**.**categories**.**shape[0]

*# get training set*

trnx\_usrs **=** trn['userId']**.**astype(ct)**.**cat**.**codes**.**values

trnx\_movies **=** trn['movieId']**.**astype(ct)**.**cat**.**codes**.**values

trn\_t **=** np**.**ones(len(trnx\_usrs))

*# add some negative samples*

u **=** np**.**random**.**randint(n\_usrs, size**=**sample\_size\_with\_no\_links)

i **=** np**.**random**.**randint(num\_of\_movies, size**=**sample\_size\_with\_no\_links)

non\_neg\_indexx **=** np**.**where(trn\_dt[u,i] **==** 0)

trnx\_usrs **=** np**.**concatenate([trnx\_usrs, u[non\_neg\_indexx[1]]])

trnx\_movies **=** np**.**concatenate([trnx\_movies, i[non\_neg\_indexx[1]]])

trn\_t **=** np**.**concatenate([trn\_t, np**.**zeros(u[non\_neg\_indexx[1]]**.**shape[0])])

print((trnx\_usrs**.**shape, trnx\_movies**.**shape, trn\_t**.**shape))

*# shuffle the data*

X **=** np**.**stack([trnx\_usrs, trnx\_movies, trn\_t], axis**=**1)

np**.**random**.**shuffle(X)

##### Model parameters

*#Output activation*

actv **=** 'sigmoid'

*#Intializer*

intt **=** 'lecun\_uniform'

*#Layer name*

nem **=** 'prediction'

*#user input*

u\_t **=** 'user\_input'

*#item input*

i\_p **=** 'item\_input'

##### Model infrastructure

*#usr\_lt = user lt*

**def** neural\_network(number\_of\_usrs, num\_movies, ld, regs**=**[0,0]):

*# model vars*

*#user\_p = user input*

usr\_p **=** Input(shape**=**(1,), dtype**=**'int64', name **=** u\_t)

itm\_input **=** Input(shape**=**(1,), dtype**=**'int64', name **=** i\_p)

*#ld = lt dimension*

*#mbds = embeddings*

*#mbds\_usrr = user embedding*

mbds\_usrr **=** Embedding(input\_dim **=** number\_of\_usrs, output\_dim **=** ld, name **=** 'user\_mbds',

embeddings\_regularizer **=** l2(regs[0]), input\_length**=**1)

*#mbds\_itm = item emdeddings*

mbds\_itm **=** Embedding(input\_dim **=** num\_movies, output\_dim **=** ld, name **=** 'itm\_mbds',

embeddings\_regularizer **=** l2(regs[1]), input\_length**=**1)

*# essential to define an embedding vector!*

usr\_lt **=** Flatten()(mbds\_usrr(usr\_p))

itm\_lt **=** Flatten()(mbds\_itm(itm\_input))

*# user-itm embedding productper element*

prdvec **=** merge**.**Multiply()([usr\_lt, itm\_lt])

*# Final recommender layer*

prd **=** Dense(1, activation**=**actv, kernel\_initializer**=**intt, name **=** nem)(prdvec)

*#return the recommendation model*

md **=** Model(inputs**=**[usr\_p, itm\_input], outputs**=**prd)

**return** md

*# create the model*

**def** rc(yt, yp):

tp **=** K**.**sum(K**.**round(K**.**clip(yt **\*** yp, 0, 1)))

pos\_pt **=** K**.**sum(K**.**round(K**.**clip(yt, 0, 1)))

recall **=** tp **/** (pos\_pt **+** K**.**epsilon())

**return** recall

mdl **=** neural\_network(n\_usrs, num\_of\_movies, 50, regs **=** [0,0])

#compile the model

mdl**.**compile(optimizer**=**RMSprop(lr**=**0.001), metrics **=** ['accuracy', rc], loss**=**'binary\_crossentropy')

early\_stop **=** EarlyStopping(monitor**=**'val\_loss', min\_delta**=**0.01, patience**=**3, verbose**=**1)

h **=** mdl**.**fit([X[:,0], X[:,1]], X[:,2], batch\_size**=**150000, epochs**=**20, validation\_split **=** 0.1, verbose**=**1, callbacks **=** [early\_stop])

score **=** mdl**.**evaluate([tstx['userId']**.**cat**.**codes**.**values, tstx['movieId']**.**cat**.**codes**.**values], np**.**ones(tstx**.**shape[0]), verbose**=**1, batch\_size**=**100000)

#Print the model performance

print(f'test Loss: {score[0]} | test Recall: {score[1]}')

Plot the loss and recall of the model across the defined epochs

*# Plot the model history*

**import** matplotlib.pyplot **as** p

fig, (axis1,axis2) **=** p**.**subplots(1,2, figsize**=**(16,4))

#Plot loss

axis1**.**plot(h**.**history['loss'])

axis1**.**plot(h**.**history['val\_loss'])

axis1**.**set\_title('Loss')

axis1**.**legend(['trn', 'tst'], loc**=**'upper left')

#plot recall

axis2**.**plot(h**.**history['rc'])

axis2**.**plot(h**.**history['val\_rc'])

axis2**.**set\_title('Model Recall Per Epoch')

axis2**.**legend(['trn', 'tst'], loc**=**'upper left')

p**.**show()

##### Evaluate the performance of the ncf model using the mpr and recall metrics

*compute MPR*

*we implement per chunk to handle memory issues*

user\_index **=** testw['userId']**.**cat**.**codes**.**values

item\_index **=** testw['movieId']**.**cat**.**codes**.**values

number\_of\_chunks **=** 10

chunks **=** np**.**array\_split(np**.**arange(n\_users), number\_of\_chunks)

res **=** []

prev\_max **=** 0

**for** i,indexx **in** enumerate(chunks):

print(f'Implementing chunk : {i**+**1}/{number\_of\_chunks}')

cross\_movieId **=** np**.**transpose([np**.**tile(np**.**arange(num\_of\_items), len(indexx)), np**.**repeat(indexx, len(np**.**arange(num\_of\_items)))])

score **=** model**.**predict([cross\_movieId[:,1], cross\_movieId[:,0]], batch\_size**=**50000, verbose**=**1)**.**reshape(indexx**.**shape[0], num\_of\_items)

score **=** (**-**score)**.**argsort() **/** score**.**shape[1]

sel **=** (user\_index **>=** indexx**.**min()) **&** (user\_index **<=** indexx**.**max())

chunk\_score **=** np**.**asarray(score[user\_index[sel] **-** prev\_max, item\_index[sel]])**.**reshape(**-**1)

res**.**append(chunk\_score)

prev\_max **=** indexx**.**max() **+** 1

Compute the mpr of the model

mpr\_N\_C\_F **=** np**.**concatenate(rs)**.**sum()**/**len(tstx)

print(f'N\_C\_F MPR: {mpr\_N\_C\_F:.5f}')

Compute recall

rc\_deep\_reac **=** (np**.**concatenate(rs) **<** 0.5)**.**sum()**/**len(tstx)

print(f'N\_C\_F Model Recall rate: {rc\_deep\_reac:.5f}')

1. Make predictions (recommendations)

*#Make movie predictions using the deep learning model*

tst\_pred['score'] **=** list(np**.**concatenate(res))

*#Select the top 5 movieIds to recommend to a user*

tsf **=** tst\_pred**.**sort\_values(['score'],ascending**=False**)**.**groupby('userId')**.**head(5)

tsf1 **=** tsf**.**sort\_values('userId')

tsf1 **=** tsf1[['userId', 'movieId']]

tsf1**.**columns **=** ['userId', 'Recommended movieId']

tsf1**.**head(10)

##### Non-Negative Matrix Factorization

**def** nmf(dt, prd,mdl):

*# normalize to obtain % significance to user*

scaled\_df **=** dt**.**multiply(1**/**dt**.**sum(axis**=**1))**.**tocsr()

S **=** mdl**.**fit\_transform(scaled\_df)

K **=** mdl**.**components\_

*# implement per bit to handle memory issues*

user\_index **=** prd['userId']**.**cat**.**codes**.**values

movie\_index **=** prd['movieId']**.**cat**.**codes**.**values

number\_of\_bits **=** 30

#apply the bits/chunks

bits **=** np**.**array\_split(np**.**arange(S**.**shape[0]), number\_of\_bits)

*#r = Results*

r **=** []

prev\_mx **=** 0

**for** i,indexx **in** enumerate(bits):

print(f'Implementing bit : {i**+**1}/{number\_of\_bits}')

scr **=** (S[indexx]**.**dot(K))

scr **=** (**-**scr)**.**argsort() **/** scr**.**shape[1]

sel **=** (user\_index **>=** indexx**.**min()) **&** (user\_index **<=** indexx**.**max())

bit\_scr **=** np**.**asarray(scr[user\_index[sel] **-** prev\_mx, movie\_index[sel]])**.**reshape(**-**1)

r**.**append(bit\_scr)

prev\_mx **=** indexx**.**max() **+** 1

**return** np**.**concatenate(r)

Kk **=** 2

*# NMF*

*#NNDSVD to handle sparsity*

*#Define the nmf model*

mdl\_nmf **=** NMF(n\_components **=** Kk, init **=** 'nndsvd', max\_iter **=** 1000, tol **=** 0.1, random\_state **=** 1)

#Generate recommendations using the nmf model

Recommendations1 **=** nmf(trn\_dt, tstx, mdl\_nmf)

mpr\_Non\_Negative\_Matrix\_Factorization **=** Recommendations1**.**sum()**/**len(tstx)

print(f'NMF MPR', round(mpr\_Non\_Negative\_Matrix\_Factorization,5))

rec\_Non\_Negative\_Matrix\_Factorization **=** (Recommendations1 **<** 0.5)**.**sum()**/**len(tstx)

#output the model performance

print('mdl Recall rate', round(rec\_Non\_Negative\_Matrix\_Factorization,5))