Book Rental Recommendation

Before reading data from a csv file, you need to download the “BX-Book-Ratings.csv”, “BX-Books.csv”, “BX-Users.csv”, and “Recommend.csv” datasets from the resource section and upload them into the Lab. We will use the Up arrow icon, which is shown on the left side under the View icon. Click on the Up arrow icon and upload the file from wherever it was downloaded into your system.

The objective is to recommend books to a user based on their purchase history and the behavior of other users.

Dataset Description

- BX-Users: It contains the information of users.
    - user_id - These have been anonymized and mapped to integers
    - Location - Demographic data is provided
    - Age - Demographic data is provided
- If available. Otherwise, these fields contain NULL-values.
- BX-Books:
    - isbn - Books are identified by their respective ISBNs. Invalid ISBNs have already been removed from the dataset.
    - book_title
    - book_author
    - year_of_publication
    - publisher
- BX-Book-Ratings: Contains the book rating information.
    - user_id
    - isbn
    - rating - Ratings (`Book-Rating`) are either explicit, expressed on a scale from 1-10 (higher values denoting higher appreciation), or implicit, expressed by 0.

Getting Started

In [1]:

```
#Import required libraries
import numpy as np
import pandas as pd
```

Read the data using pandas in DataFrame df_user

To map byte values directly to the first 256 Unicode code points, use the "Latin-1" encoding. This is the closest equivalent Python 3 offers to the permissive Python 2 text handling model.

In [2]:

```
#Reading the data
df_user = pd.read_csv('BX-Users.csv',encoding='latin-1',low_memory=False)
```

Preview the information of first 5 rows of dataset.

In [3]:

```
df_user.head(6)
```

Out[3]:

	user_id	Location	Age
0	1	nyc, new york, usa	NaN
1	2	stockton, california, usa	18.0
2	3	moscow, yukon territory, russia	NaN
3	4	porto, v.n.gaia, portugal	17.0
4	5	farnborough, hants, united kingdom	NaN
5	6	santa monica, california, usa	61.0

Checking for Null Values

In [4]:

```
df_user.isnull().any()
```

Out[4]:

```
user_id     False
Location     True
Age          True
dtype: bool
```

Dropping the Null Values

In [5]:

```
df_user1=df_user.dropna()
```

In [6]:

```
df_user1.isnull().any()
```

Out[6]:

```
user_id     False
Location    False
Age         False
dtype: bool
```

Read the books Data and explore

In [27]:

```
df_books = pd.read_csv('BX-Books.csv', encoding='latin-1',low_memory=False)
```

Preview the information of first 5 rows of dataset.

In [8]:

```
df_books.head()
```

Out[8]:

	isbn	book_title	book_author	year_of_publication	publisher
0	195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press
1	2005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
2	60973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial
3	374157065	Flu: The Story of the Great Influenza Pandemic...	Gina Bari Kolata	1999	Farrar Straus Giroux
4	393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton &amp; Company

Reading the data where ratings are given

You will read only first 10000 rows otherwise, Out Of Memory error can occur.

In [9]:

```
df = pd.read_csv('BX-Book-Ratings.csv',encoding='latin-1',nrows=10000)
```

Preview the information of first 5 rows of dataset.

In [10]:

```
df.head()
```

Out[10]:

	user_id	isbn	rating
0	276725	034545104X	0
1	276726	155061224	5
2	276727	446520802	0
3	276729	052165615X	3
4	276729	521795028	6

Using '**describe()**' function

It is used to view some basic statistical details like percentile, mean, std.

In [11]:

```
df.describe()
```

Out[11]:

	user_id	rating
count	10000.000000	10000.000000
mean	265844.379600	1.974700
std	56937.189618	3.424884
min	2.000000	0.000000
25%	277478.000000	0.000000
50%	278418.000000	0.000000
75%	278418.000000	4.000000
max	278854.000000	10.000000

Merge the dataframes.

For all practical purposes, User Master Data is not required. So, ignore dataframe df_user

In [12]:

```
df = pd.merge(df,df_books,on='isbn')
df.head()
```

Out[12]:

	user_id	isbn	rating	book_title	book_author	year_of_publication	publisher
0	276725	034545104X	0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books
1	276726	155061224	5	Rites of Passage	Judith Rae	2001	Heinle
2	276727	446520802	0	The Notebook	Nicholas Sparks	1996	Warner Books
3	278418	446520802	0	The Notebook	Nicholas Sparks	1996	Warner Books
4	276729	052165615X	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press

Checking for unique users and books

Here we are using **'nunique()'** function that returns the Series with the number of distinct observations over the requested axis.

In [13]:

```
#Code for checking number of unique users and books.
n_users = df.user_id.nunique()
n_books = df.isbn.nunique()

print('Num. of Users: '+ str(n_users))
print('Num of Books: '+str(n_books))
```

```
Num. of Users: 828
Num of Books: 8051
```

Convert ISBN variable to numeric type in order

In [14]:

```
#Convert and print length of isbn list
isbn_list = df.isbn.unique()
print(" Length of isbn List:", len(isbn_list))
def get_isbn_numeric_id(isbn):
    #print ("  isbn is:" , isbn)
    itemindex = np.where(isbn_list==isbn)
    return itemindex[0][0]
```

```
 Length of isbn List: 8051
```

Convert user_id variable to numeric type in order

In [15]:

```
#Convert and print length of user_id list
userid_list = df.user_id.unique()
print(" Length of user_id List:", len(userid_list))
def get_user_id_numeric_id(user_id):
    #print ("  isbn is:" , isbn)
    itemindex = np.where(userid_list==user_id)
    return itemindex[0][0]
```

```
 Length of user_id List: 828
```

Convert both user_id and isbn to ordered list i.e. from 0...n-1

In [16]:

```
df['user_id_order'] = df['user_id'].apply(get_user_id_numeric_id)
```

In [17]:

```
df['isbn_id'] = df['isbn'].apply(get_isbn_numeric_id)
df.head()
```

Out[17]:

	user_id	isbn	rating	book_title	book_author	year_of_publication	publisher	user_id_order	isbn_id
0	276725	034545104X	0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	0	0
1	276726	155061224	5	Rites of Passage	Judith Rae	2001	Heinle	1	1
2	276727	446520802	0	The Notebook	Nicholas Sparks	1996	Warner Books	2	2
3	278418	446520802	0	The Notebook	Nicholas Sparks	1996	Warner Books	3	2

4	276729	052165615X	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press	4	3

Re-index columns to build matrix

In [18]:

```
#Reindexing the columns
new_col_order = ['user_id_order', 'isbn_id', 'rating', 'book_title', 'book_author','year_of_publication','publi
df = df.reindex(columns= new_col_order)
df.head()
```

Out[18]:

	user_id_order	isbn_id	rating	book_title	book_author	year_of_publication	publisher	isbn	user_id
0	0	0	0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	034545104X	276725
1	1	1	5	Rites of Passage	Judith Rae	2001	Heinle	155061224	276726
2	2	2	0	The Notebook	Nicholas Sparks	1996	Warner Books	446520802	276727
3	3	2	0	The Notebook	Nicholas Sparks	1996	Warner Books	446520802	278418
4	4	3	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press	052165615X	276729

Train Test Split

Recommendation Systems are difficult to evaluate, but you will still learn how to evaluate them. In order to do this, you'll split your data into two sets. However, you won't do your classic X_train,X_test,y_train,y_test split. Instead, you can actually just segement the data into two sets of data:

Importing train_test_split model

In [19]:

```
#Importing train_test_split model for splittig the data into train and test set
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(df, test_size=0.30)
```

Approach: You Will Use Memory-Based Collaborative Filtering

Memory-Based Collaborative Filtering approaches can be divided into two main sections: **user-item filtering** and **item-item filtering**.

A *user-item filtering* will take a particular user, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked.

In contrast, *item-item filtering* will take an item, find users who liked that item, and find other items that those users or similar users also liked. It takes items as input and outputs other items as recommendations.

- *Item-Item Collaborative Filtering*: “Users who liked this item also liked …”
- *User-Item Collaborative Filtering*: “Users who are similar to you also liked …”

In both cases, you create a user-book matrix which is built from the entire dataset.

Since you have split the data into testing and training, you will need to create two `[828 x 8051]` matrices (all users by all books). This is going to be a very large matrix

The training matrix contains 70% of the ratings and the testing matrix contains 30% of the ratings.

Create two user-book matrix for training and testing

> Indented block

In [20]:

```
#Create user-book matrix for training
train_data_matrix = np.zeros((n_users, n_books))
for line in train_data.itertuples():
    train_data_matrix[line[1]-1, line[2]-1] = line[3]

#Create user-book matrix for testing
test_data_matrix = np.zeros((n_users, n_books))
for line in test_data.itertuples():
    test_data_matrix[line[1]-1, line[2]-1] = line[3]
```

Import Pairwise Model

You can use the pairwise_distances function from sklearn to calculate the cosine similarity. Note, the output will range from 0 to 1 since the ratings are all positive.

In [21]:

```
#Importing pairwise_distances function
from sklearn.metrics.pairwise import pairwise_distances
user_similarity = pairwise_distances(train_data_matrix, metric='cosine')
item_similarity = pairwise_distances(train_data_matrix.T, metric='cosine')
```

In [22]:

```
user_similarity
```

Out[22]:

```
array([[0., 1., 1., ..., 1., 1., 1.],
       [1., 0., 1., ..., 1., 1., 1.],
       [1., 1., 0., ..., 1., 1., 1.],
       ...,
       [1., 1., 1., ..., 0., 1., 1.],
       [1., 1., 1., ..., 1., 0., 1.],
       [1., 1., 1., ..., 1., 1., 0.]])
```

Make predictions

In [23]:

```
#Defining custom function to make predictions
def predict(ratings, similarity, type='user'):
    if type == 'user':
        mean_user_rating = ratings.mean(axis=1)
        #You use np.newaxis so that mean_user_rating has same format as ratings
        ratings_diff = (ratings - mean_user_rating[:, np.newaxis])
        pred = mean_user_rating[:, np.newaxis] + similarity.dot(ratings_diff) / np.array([np.abs(similarity).su
    elif type == 'item':
        pred = ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])
    return pred
```

In [24]:

```
item_prediction = predict(train_data_matrix, item_similarity, type='item')
user_prediction = predict(train_data_matrix, user_similarity, type='user')
```

Evaluation

There are many evaluation metrics, but one of the most popular metric used to evaluate accuracy of predicted ratings is *Root Mean Squared Error (RMSE)*.

Since, you only want to consider predicted ratings that are in the test dataset, you filter out all other elements in the prediction matrix with: `prediction[ground_truth.nonzero()]`.

In [25]:

```
#Importing RMSE function
from sklearn.metrics import mean_squared_error
from math import sqrt

#Defining custom function to filter out elements with ground_truth.nonzero
def rmse(prediction, ground_truth):
    prediction = prediction[ground_truth.nonzero()].flatten()
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    return sqrt(mean_squared_error(prediction, ground_truth))
```

Printing RMSE value for user based and item based collaborative filtering

In [26]:

```
print('User-based CF RMSE: ' + str(rmse(user_prediction, test_data_matrix)))
print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))
```

```
User-based CF RMSE: 7.679518755872175
Item-based CF RMSE: 7.679029331453672
```

Both the approach yield almost same result

End