simplilearn

Assignment: Book Rental Recommendation

The comments/sections provided are your cues to perform the assignment. You don't need to limit yourself to the number of rows/cells provided. You can add additional rows in each section to add more lines of code.

If at any point in time you need help on solving this assignment, view our demo video to understand the different steps of the code.

Happy coding!

Book Rental Recommendation

DESCRIPTION

BookRent is the largest online and offline book rental chain in India. The company charges a fixed rental fee for a book per month. Lately, the company has been losing its user base. The main reason for this is that users are not able to choose the right books for themselves. The company wants to solve this problem and increase its revenue and profit.

Objective: You, as an ML expert, have to model a recommendation engine so that users get recommendations for books based on the behavior of similar users. This will ensure that users are renting books based on their individual tastes.

Actions to Perform:

- Read the books dataset and explore it.
- Clean up NaN values.
- Read the data where ratings are given by users.
- Take a quick look at the number of unique users and books.
- Convert ISBN to numeric numbers in the correct order.
- Do the same for user_id. Convert it into numeric order.
- Convert both user_id and ISBN to the ordered list i.e. from 0...n-1.
- Re-index columns to build matrix later on.
- Split your data into two sets (training and testing).
- Calculate the cosine similarity.
- Use the evaluation metrics to make predictions.

In [1]:
```python
#Import required libraries
import numpy as np
import pandas as pd
import seaborn as sns
```

Read the datasets and explore it

In [2]:
```python
#Read the data using pandas in DataFrame
df_users = pd.read_csv('BX-Users.csv',encoding='latin-1')
df_users.head()
```

```
C:\Users\Sohaib\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3165: DtypeWarning: Columns (0) have mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[2]:

	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

In [3]:
```python
df_users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278859 entries, 0 to 278858
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype  
---  ------    --------------   -----  
 0   user_id   278859 non-null  object 
 1   Location  278858 non-null  object 
 2   Age       168096 non-null  float64
dtypes: float64(1), object(2)
memory usage: 4.3+ MB
```

The Age column in df_users dataset has number of NaN values, which can be dropped. since we have a large set of dataset (168096 rows) with non-null values in Age column. we can drop entire set of rows with NaN values.

In [4]:
```python
df_users.shape
```

Out[4]: `(278859, 3)`

In [5]:
```python
#Read the books Data and explore
#column_names = ['isbn', 'book_title']
df_books = pd.read_csv('BX-Books.csv', encoding='latin-1')
df_books.head()
```

```
C:\Users\Sohaib\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3165: DtypeWarning: Columns (3) have mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[5]:

	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 & Company

In [6]:
```python
df_books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271379 entries, 0 to 271378
Data columns (total 5 columns):
 #   Column               Non-Null Count   Dtype 
---  ------               --------------   ----- 
 0   isbn                 271379 non-null  object
 1   book_title           271379 non-null  object
 2   book_author          271378 non-null  object
 3   year_of_publication  271379 non-null  object
 4   publisher            271377 non-null  object
dtypes: object(5)
memory usage: 5.2+ MB
```

The book_author and publisher columns in df_books dataset has number of NaN values. Since we have a large set of dataset NaN rows can be dropped. .

In [7]:
```python
df_books.shape
```

Out[7]: `(271379, 5)`

Read the data where ratings are given by users

We have taken only first 5000 rows of ratings given by users. Otherwise, Out Of Memory error can occur.

In [8]:
```python
df_ratings = pd.read_csv('BX-Book-Ratings.csv',encoding='latin-1',nrows=5000)
df_ratings.head()
```

Out[8]:

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

In [9]:
```python
df_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   user_id  5000 non-null   int64 
 1   isbn     5000 non-null   object
 2   rating   5000 non-null   int64 
dtypes: int64(2), object(1)
memory usage: 97.7+ KB
```

Clean up NaN values

Dropping rows with NaN values in both df_users and df_books DataFrames.

In [10]:
```python
#Clean up NaN values in df_users
df_users = df_users.dropna(subset=['Age'])
df_users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 168096 entries, 1 to 278855
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype  
---  ------    --------------   -----  
 0   user_id   168096 non-null  object 
 1   Location  168096 non-null  object 
 2   Age       168096 non-null  float64
dtypes: float64(1), object(2)
memory usage: 3.8+ MB
```

In [11]:
```python
#Clean up NaN values in df_books
df_books = df_books.dropna(subset=['book_author','publisher'])
df_books.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 271376 entries, 0 to 271378
Data columns (total 5 columns):
 #   Column               Non-Null Count   Dtype 
---  ------               --------------   ----- 
 0   isbn                 271376 non-null  object
 1   book_title           271376 non-null  object
 2   book_author          271376 non-null  object
 3   year_of_publication  271376 non-null  object
 4   publisher            271376 non-null  object
dtypes: object(5)
memory usage: 7.2+ MB
```

Merging the df_ratings and df_books dataframes based on 'isbn', to form a new combined DataFrame 'df'.

In [12]:
```python
df = pd.merge(df_ratings,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	276729	052165615X	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press
4	276729	521795028	6	The Amsterdam Connection : Level 4 (Cambridge ...	Sue Leather	2001	Cambridge University Press

In [13]:
```python
df.shape
```

Out[13]: `(4427, 7)`

Take a quick look at the number of unique users and books

In [14]:
```python
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: 591
Num of Books: 4122
```

Convert ISBN to numeric numbers in the correct order

In [15]:
```python
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: 4122
```

Convert user_id to numeric numbers in the correct order

In [16]:
```python
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: 591
```

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

In [17]:
```python
df['user_id_order'] = df['user_id'].apply(get_user_id_numeric_id)
```

In [18]:
```python
df['isbn_id'] = df['isbn'].apply(get_isbn_numeric_id)
df.head()
```

Out[18]:

	user_id	isbn	rating	book_title	book_author	year_of_publication	publisher	user_id_order	isbn_i
0	276725	034545104X	0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	0	
1	276726	155061224	5	Rites of Passage	Judith Rae	2001	Heinle	1	
2	276727	446520802	0	The Notebook	Nicholas Sparks	1996	Warner Books	2	
3	276729	052165615X	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press	3	
4	276729	521795028	6	The Amsterdam Connection : Level 4 (Cambridge ...	Sue Leather	2001	Cambridge University Press	3	

Re-index columns to build matrix later on

In [19]:
```python
new_col_order = ['user_id_order', 'isbn_id', 'rating', 'book_title', 'book_author',
                 'year_of_publication','publisher','isbn','user_id']
df = df.reindex(columns= new_col_order)
df.head()
```

Out[19]:

	user_id_order	isbn_id	rating	book_title	book_author	year_of_publication	publisher	isbn	user_i
0	0	0	0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	034545104X	27672
1	1	1	5	Rites of Passage	Judith Rae	2001	Heinle	155061224	27672
2	2	2	0	The Notebook	Nicholas Sparks	1996	Warner Books	446520802	27672
3	3	3	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press	052165615X	27672
4	3	4	6	The Amsterdam Connection : Level 4 (Cambridge ...	Sue Leather	2001	Cambridge University Press	521795028	27672

Split data into two sets (training and testing)

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

In [20]:
```python
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(df, test_size=0.30)
```

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, we create a user-book matrix which is built from the entire dataset.

Since we have split the data into testing and training, we 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.

In [21]:
```python
#Create two user-book matrices, one for training and another for testing
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]

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]
```

Calculate the cosine similarity.

we 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 [22]:
```python
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 [23]:
```python
user_similarity
```

Out[23]:
```
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.]])
```

In [24]:
```python
item_similarity
```

Out[24]:
```
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.]])
```

Use the evaluation metrics to make predictions

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)*.

In [25]:
```python
def predict(ratings, similarity, type='user'):
    if type == 'user':
        mean_user_rating = ratings.mean(axis=1)
        #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.ar

    elif type == 'item':
        pred = ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])
    return pred
```

In [26]:
```python
item_prediction = predict(train_data_matrix, item_similarity, type='item')
user_prediction = predict(train_data_matrix, user_similarity, type='user')
```

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

In [27]:
```python
from sklearn.metrics import mean_squared_error
from math import sqrt
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))
```

In [28]:
```python
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: 7.725507960248443
Item-based CF RMSE: 7.722860869362833
```

From the above results, we can infer that the both User-based and Item-based approach yield almost same results.