book-rental-recommendation

March 24, 2023

Project 3 - Book Rental Recommendation

DESCRIPTION

Book Rent is the largest online and offline book rental chain in India. They provide books of various genres, such as thrillers, mysteries, romances, and science fiction. 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.

Project Objective:

You, as an ML expert, should focus on improving the user experience by personalizing it to the user's needs. You 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 the books based on their tastes and traits.

Note: You have to perform user-based collaborative filtering and item-based collaborative filtering.

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.

Note: Download the "BX-Book-Ratings.csv", "BX-Books.csv", "BX-Users.csv", and "Recommend.csv" using the link given in the Book Rental Recommendation project problem statement.

Following operations should be performed:

- 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 variables to numeric numbers in the correct order
- Convert the user_id variable to numeric numbers in the correct order
- Convert both user_id and ISBN to the ordered list, i.e., from 0…n-1
- Re-index the columns to build a matrix
- Split your data into two sets (training and testing)
- Make predictions based on user and item variables
- Use RMSE to evaluate the predictions

[1]:
```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')
```

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.*

[2]:
```
df_user = pd.read_csv('BX-Users.csv',encoding='latin-1')
```

[3]:
```
df_user.head()
```

[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
```

[4]:
```
df_user.tail()
```

[4]:
```
        user_id                             Location   Age
278854   278854                portland, oregon, usa   NaN
278855   278855  tacoma, washington, united kingdom  50.0
278856   278856             brampton, ontario, canada   NaN
278857   278857             knoxville, tennessee, usa   NaN
278858   278858                  dublin, n/a, ireland   NaN
```

[5]:
```
df_user.shape
```

[5]:
```
(278859, 3)
```

[6]:
```
df_user.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: 6.4+ MB
```

[7]:
```
# Checking for Null Values.
df_user.isnull().sum()
```

[7]:
```
user_id          0
Location         1
Age         110763
dtype: int64
```

[8]:
```
df_user.isnull().any()
```

[8]:
```
user_id     False
Location     True
Age          True
dtype: bool
```

[9]:
```
# Dropping the Null Values.
df_user1=df_user.dropna()
```

[10]:
```
df_user1.isnull().sum()
```

[10]:
```
user_id     0
Location    0
Age         0
dtype: int64
```

[11]:
```
df_user1.isnull().any()
```

[11]:
```
user_id     False
Location    False
Age         False
dtype: bool
```

Read the books Data and explore

[12]:
```
df_books = pd.read_csv('BX-Books.csv', encoding='latin-1')
```

[13]:
```
df_books.head()
```

```
[13]:         isbn                                          book_title  \
      0  195153448                                 Classical Mythology
      1    2005018                                        Clara Callan
      2   60973129                                Decision in Normandy
      3  374157065  Flu: The Story of the Great Influenza Pandemic…
      4  393045218                              The Mummies of Urumchi

                  book_author year_of_publication                   publisher
      0    Mark P. O. Morford                2002     Oxford University Press
      1  Richard Bruce Wright                2001       HarperFlamingo Canada
      2          Carlo D'Este                1991             HarperPerennial
      3      Gina Bari Kolata                1999        Farrar Straus Giroux
      4       E. J. W. Barber                1999  W. W. Norton &amp; Company
```

```
[14]: df_books.shape
```

```
[14]: (271379, 5)
```

Reading the data where ratings are given *We will read only first 10000 rows otherwise, Out Of Memory error can occur.*

```
[15]: df_ratings = pd.read_csv('BX-Book-Ratings.csv',encoding='latin-1',nrows=10000)
```

```
[16]: df_ratings.head()
```

```
[16]:    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.*

```
[17]: df_ratings.describe()
```

```
[17]:              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*

```
[18]: df_final = pd.merge(df_ratings,df_books,on='isbn')
```

```
[19]: df_final.head()
```

```
[19]:    user_id        isbn  rating             book_title      book_author  \
      0   276725  034545104X       0  Flesh Tones: A Novel       M. J. Rose
      1   276726   155061224       5      Rites of Passage       Judith Rae
      2   276727   446520802       0          The Notebook  Nicholas Sparks
      3   278418   446520802       0          The Notebook  Nicholas Sparks
      4   276729  052165615X       3        Help!: Level 1    Philip Prowse

        year_of_publication                   publisher
      0                2002            Ballantine Books
      1                2001                      Heinle
      2                1996                Warner Books
      3                1996                Warner Books
      4                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.

```
[20]: # Code for checking number of unique users and books.
      n_users = df_final.user_id.nunique()
      n_books = df_final.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

```
[21]: # Convert and print length of isbn list
      isbn_list = df_final.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 *This is formatted as code.*

```
[22]: # Convert and print length of user_id list
      userid_list = df_final.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

[23]:
```
df_final['user_id_order'] = df_final['user_id'].apply(get_user_id_numeric_id)
```

[24]:
```
df_final['isbn_id'] = df_final['isbn'].apply(get_isbn_numeric_id)
df_final.head()
```

[24]:
```
   user_id        isbn  rating            book_title      book_author  \
0   276725  034545104X       0  Flesh Tones: A Novel       M. J. Rose
1   276726   155061224       5      Rites of Passage       Judith Rae
2   276727   446520802       0          The Notebook  Nicholas Sparks
3   278418   446520802       0          The Notebook  Nicholas Sparks
4   276729  052165615X       3        Help!: Level 1    Philip Prowse

  year_of_publication                   publisher  user_id_order  isbn_id
0                2002            Ballantine Books              0        0
1                2001                      Heinle              1        1
2                1996                Warner Books              2        2
3                1996                Warner Books              3        2
4                1999  Cambridge University Press              4        3
```

Re-index columns to build matrix

[25]:
```
# Reindexing the columns
new_col_order = ['user_id_order', 'isbn_id', 'rating', 'book_title',␣
↪'book_author','year_of_publication','publisher','isbn','user_id']
df_final = df_final.reindex(columns= new_col_order)
df_final.head()
```

[25]:
```
   user_id_order  isbn_id  rating            book_title      book_author  \
0              0        0       0  Flesh Tones: A Novel       M. J. Rose
1              1        1       5      Rites of Passage       Judith Rae
2              2        2       0          The Notebook  Nicholas Sparks
3              3        2       0          The Notebook  Nicholas Sparks
4              4        3       3        Help!: Level 1    Philip Prowse

  year_of_publication                   publisher        isbn  user_id
0                2002            Ballantine Books  034545104X   276725
1                2001                      Heinle   155061224   276726
2                1996                Warner Books   446520802   276727
```

```
3                 1996                Warner Books   446520802   278418
4                 1999  Cambridge University Press  052165615X   276729
```

Train Test Split

Recommendation Systems are difficult to evaluate, but we will still learn how to evaluate them. In order to do this, will 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:

Importing train_test_split model

```
[26]: # 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_final, test_size=0.20)
```

Approach: We 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, we will 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 80% of the ratings and the testing matrix contains 20% of the ratings.

Create two user-book matrix for training and testing

Indented block

```
[27]: # 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 *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.*

[28]:
```
# 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')
```

[29]:
```
user_similarity
```

[29]:
```
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

[30]:
```
# Defining custom function to make predictions
def predict(ratings, similarity, type='user'):
    if type == 'user':
        mean_user_rating = ratings.mean(axis=1)
        # We will 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).sum(axis=1)]).T
    elif type == 'item':
        pred = ratings.dot(similarity) / np.array([np.abs(similarity).
 ↪sum(axis=1)])
    return pred
```

[31]:
```
item_prediction = predict(train_data_matrix, item_similarity, type='item')
user_prediction = predict(train_data_matrix, user_similarity, type='user')
```

[32]:
```
print(item_prediction)
```

```
[[0.         0.00062112 0.00062112 … 0.00062159 0.00062112 0.00062112]
 [0.         0.         0.         … 0.         0.         0.        ]
 [0.06571429 0.06571429 0.06571429 … 0.06576407 0.06571429 0.06571429]
 …
 [0.         0.         0.         … 0.         0.         0.        ]
 [0.         0.         0.         … 0.         0.         0.        ]
 [0.         0.         0.         … 0.         0.         0.        ]]
```

[33]:
```
print(user_prediction)
```

```
[[-0.00135983 -0.00135983  0.00226774 …  0.00952288 -0.00135983
  -0.00135983]
 [ 0.00406433 -0.00198162  0.00164595 …  0.00890109 -0.00198162
  -0.00198162]
 [ 0.06985073  0.06380399  0.06743203 …  0.07468811  0.06380399
   0.06380399]
 …
 [ 0.00406433 -0.00198162  0.00164595 …  0.00890109 -0.00198162
  -0.00198162]
 [ 0.00406433 -0.00198162  0.00164595 …  0.00890109 -0.00198162
  -0.00198162]
 [ 0.00406433 -0.00198162  0.00164595 …  0.00890109 -0.00198162
  -0.00198162]]
```

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

[34]:
```python
# 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

[35]:
```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: 7.663374330427166
Item-based CF RMSE: 7.662434131747383
```

Both the approach yield almost same result.