Recommender System - Recommend Books

Recommendation Systems usually rely on larger datasets and specifically need to be organized in a particular fashion.

Objective is to recommend books to a user based on purchase history and behavior of other users

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

In [2]:
```
df = pd.read_csv("book_ratings.csv")
df.head()
```

Out[2]:

	user_id_order	isbn_id	rating	book_title	book_author	year_of_publication	publisher	is
0	0	0	0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	0345451(
1	1	1	5	Rites of Passage	Judith Rae	2001	Heinle	155061
2	2	2	0	The Notebook	Nicholas Sparks	1996	Warner Books	446520
3	3	2	0	The Notebook	Nicholas Sparks	1996	Warner Books	446520
4	4	3	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press	0521656

In [3]:
```
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
```

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:

In [4]:
```
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.

In [5]:
```
#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]
```

You can use the pairwise_distances (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances.html) function from sklearn to calculate the cosine similarity. Note, the output will range from 0 to 1 since the ratings are all positive.

In [6]:
```
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')
```

Next step is to make predictions

In [7]:
```
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) / r
    elif type == 'item':
        pred = ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)
    return pred
```

In [8]:
```
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 [9]:
```
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 [10]:
```
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: 7.7437772349132326
Item-based CF RMSE: 7.743181380082656
```

Both the approach yield almost same result