Week 4 - Recommender System Project

Project Description

The project will use a recommender system to find items on an ecommerce site. Data consists of two files:

- File 1: csv list of comma-delimited Item ID, User ID
- File 2: csv list of comma-delimited Item ID, Item ID converted, Item Description

In [10]:
```
# Import libraries
import pandas as pd
from collections import defaultdict
```

Load data

In [11]:
```
# @hidden_cell
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0


# The following code accesses a file in your IBM Cloud Object Storage. It incl
udes your credentials.
# You might want to remove those credentials before you share the notebook.

# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__,
body )
```

In [12]:
```
# File 1
df = pd.read_csv(body)
df.head()
```

Out[12]:

	A	b
0	1000-1000-10-7058RITZSM	1000-1000-100151-SO
1	1000-1000-10-7058RITZSM	1000-1000-105558-SO
2	1000-1000-10-7058RITZSM	1000-1000-105558-SO
3	1000-1000-10-19232565	1000-1000-124207-SO
4	1000-1000-10-1077046	1000-1000-124207-SO

In [13]:
```
# @hidden_cell
```

In [14]:
```
# File 2
dataset = pd.read_csv(body1, delimiter='|', encoding='cp1252',error_bad_lines=
False)
dataset.head()
```

```
b'Skipping line 38547: expected 3 fields, saw 5\nSkipping line 66436: expecte
d 3 fields, saw 5\nSkipping line 66437: expected 3 fields, saw 5\nSkipping li
ne 70464: expected 3 fields, saw 5\nSkipping line 122890: expected 3 fields,
saw 5\nSkipping line 150768: expected 3 fields, saw 5\nSkipping line 150769:
expected 3 fields, saw 5\nSkipping line 154768: expected 3 fields, saw 5\n'
```

Out[14]:

	a	b	c
0	0001-0001-01-ENVERDEFAMILYSS	ENVERDEFAMILYSS	Enverde Family Sell Sheets
1	0001-0001-01-ENVERDENATURALSS	ENVERDENATURALSS	Enverde Natural Sell Sheets
2	1000-1000-10-1-C PCH COFFEEMKR	1-C PCH COFFEEMKR	1-Cup Pouch Coffeemakers
3	1000-1000-10-1-C POD COFFEEMKR	1-C POD COFFEEMKR	1-Cup Pod Coffeemakers
4	1000-1000-10-1-CUP DECAF COFFE	1-CUP DECAF COFFEE	1-Cup Decaf Coffee

Clean Duplicates

In [15]:
```
df_clean=df.drop_duplicates()
df_clean.shape
```

Out[15]: (4149403, 2)

Setup dictionaries for item and user

In [16]:
```
usersPerItem=defaultdict(set)
itemsPerUsers = defaultdict(set)
itemNames={}
```

In [17]:
```
df_clean[0:1]
```

Out[17]:

	A	b
0	1000-1000-10-7058RITZSM	1000-1000-100151-SO

Build the dictionaries

In [18]:
```
for user, item in zip(df_clean['b'], df_clean['A']):
    usersPerItem[item].add(user)
    itemsPerUsers[user].add(item)
```

Define functions for recommending items

In [80]:
```
def Jacard(s1,s2):
    numer=len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer/denom

def in_list(item,L):
    #print(type(L))
    for i in L:
        if item in i:
            return L.index(i)
    return -1
```

In [76]:
```
def most_similar(i):
    similarities=[]
    runningList=[]
    users =usersPerItem[i]
    candidateProducts= set()
    for u in users:
        candidateProducts = candidateProducts.union(itemsPerUsers[u])
        for i2 in candidateProducts:
            if(i2==i): continue
            sim =Jacard(users, usersPerItem[i2])
            if len(runningList) == 0:
                runningList.append(str(i2))
                similarities.append((sim,str(i2)))
            else:
                if in_list(str(i2),runningList) == -1:
                    similarities.append((sim,str(i2)))
                    runningList.append(str(i2))


    similarities.sort(reverse=True)
    return similarities[:10]
```

Select a random item and return top 10 recommended items

In [93]:
```
query = df_clean.loc[1000,'A']
match = dataset[dataset['a']==query]

#print(query , ":" , match.iloc[0][2])
```

```
Mattress Topper, Queen, 24oz
```

In [94]:
```
# convert the lookup dataframe to a list
listOfItems = dataset.values.tolist()
print(listOfItems[0])
```

```
['0001-0001-01-ENVERDEFAMILYSS', 'ENVERDEFAMILYSS', 'Enverde Family Sell Shee
ts']
```

In [ ]:
```
# run the code to find matches
```

In [ ]:
```
matches = most_similar(query)
#print(matches)
finallist=[]
for score, item in zip(matches[0], matches[1]):
    match = [d[2] for d in listOfItems if str(d[0])==item]
    itemDesc=item
    try:
        itemDesc = match[0]
    except:
        itemDesc=item

    finallist.append((score,item,itemDesc))
```

In [92]:
```
dffinal = pd.DataFrame (finallist).transpose()
dffinal
```

Out[92]:

	0	1
0	0.864407	1000-1000-10-7058RITZSK
1	0.836364	1000-1000-10-1071706
2	0.836364	Asprey, Shoe Shine Sponge

In [ ]:
```
```