In [ ]:
```
# Tom, Jerry, Pooja
# import xxx
```

In [ ]:
```
# !gdown --id '16YwkNw726Dzoel9nj3USQI8IKpHdpLR0' --output articles.csv
!gdown --id '1LHu5i9UsQu1QkwIuFHlbU1BO4Ch_7D0I' --output articles_sample.csv
# !gdown --id '1prahVQjRdMPJi6jV4cPZBGALLE6_NhSF' --output customers.csv
!gdown --id '1z9HD7ehxsSotZLmM4P-_OmGBHjKshfiR' --output customers_sample.csv
!gdown --id '1ZMbwK6JO_E9rwkptU6O-unpNY-bGSQS3' --output transactions_sample.csv

!gdown --id '1J6yh-0NujpPv0xVIvhDFzP7yt4Ao6xww' --output m1_train.csv
!gdown --id '1XA5djU7iLUZBzrgM0-SdADoTkPbvi7_h' --output m1_test.csv
```

Data Preprocessing

In [4]:
```
import pandas as pd
import numpy as np
articles = pd.read_csv('articles_sample.csv',dtype={'article_id': str})
```

Method 1: Collaborative Filtering

a. Matrix Factorization

Data Import

In [16]:
```
# HYPER PARAMS
TEST_SIZE = 20
```

In [5]:
```
import pandas as pd
import numpy as np
import scipy

m1_articles = pd.read_csv('articles_sample.csv')
m1_customers = pd.read_csv('customers_sample.csv')
m1_trans = pd.read_csv('transactions_sample.csv')
```

```
# before remove
print('Size before remove')
print('articles', len(m1_articles))
print('customers', len(m1_customers))
print('transactions', len(m1_trans))


# filtered_customer_ids: customer ids with >=2 transactions
filtered_customer_ids = m1_trans.groupby(['customer_id'])['customer_id'].count()
filtered_customer_ids = filtered_customer_ids[filtered_customer_ids >= 2].index.tolist()

# to_remove_customer_ids: customer ids with <2 trans
to_remove_customer_ids = m1_customers[~m1_customers['customer_id'].isin(filtered_customer_ids)]
to_remove_customer_ids = to_remove_customer_ids['customer_id'].values.tolist()

# remove customers from to_remove_customer_ids
m1_customers = m1_customers[~m1_customers['customer_id'].isin(to_remove_customer_ids)]
m1_trans = m1_trans[~m1_trans['customer_id'].isin(to_remove_customer_ids)]

print('\nSize after remove:')
print('articles', len(m1_articles))
print('customers', len(m1_customers))
print('transactions', len(m1_trans))
```

```
Size before remove
articles 51894
customers 13720
transactions 316443

Size after remove:
articles 51894
customers 12306
transactions 315112
```

Train Test Split

This section takes dozens of minutes to run. To reduce runtime, please import 'm1_train.csv' and 'm1_test.csv' instead.

Option 1:

In [ ]:
```
import math
```

```
customer_to_trans_num = m1_trans.groupby(['customer_id'])['customer_id'].count().to_dict()
m1_train = pd.DataFrame(columns=m1_trans.columns)
m1_test = pd.DataFrame(columns=m1_trans.columns)

counter = 0
for key in customer_to_trans_num.keys():
  if counter % 1000 == 0:
    print(counter)
  num_of_training = math.floor(customer_to_trans_num[key] * 0.8)
  num_of_testing = customer_to_trans_num[key] - num_of_training
  if num_of_testing == 0:
    num_of_testing = 1
    num_of_training -= 1
  trans = m1_trans.loc[m1_trans['customer_id'] == key]
  for index, row in trans.iterrows():
    if num_of_training > 0:
      m1_train.loc[len(m1_train.index)] = row
      num_of_training -= 1
    elif num_of_testing > 0:
      m1_test.loc[len(m1_test.index)] = row
      num_of_testing -= 1
  counter += 1

print(len(m1_train), len(m1_test), len(m1_train)/(len(m1_train)+len(m1_test)))
m1_train.to_csv('m1_train.csv')
m1_test.to_csv('m1_test.csv')
```

Option 2

In [6]:
```
m1_train = pd.read_csv('m1_train.csv')
m1_test = pd.read_csv('m1_test.csv')
```

Data Training

In [7]:
```
# create pivot table for the training set
m1_train['purchased'] = 1

df = m1_train.pivot_table(index = 'customer_id', columns ='article_id', values = 'purchased').fillna(0)
df
```

Out[7]:

article_id customer_id	108775015	108775044	108775051	110065001	110065002	110065011	111565001	1115
000362878a3904e1fe4927bbfcdb10c64a9d85b12a593a7d9c965efcc9187cbf	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
001270f9c6b9e0fe78121ca9d9071f4b9078da72258a7aed35bdaf125a65b6d0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
0019c05146d30111b7003700c712f9354fb62f9b87e53b1c9cc9f0d22fb5c62b	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
00227494dd4e87da02bb1ab4afc38f13f2e11c6517b1bbcdca56f212a9c470ab	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
0036414a83662a899a2e2c30854d8224932cc2e0d6f9c5534962a33cbc692d74	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
ffd8a6ce04a08854dc847409f7b9f1506f4509482ef73c89bf98421859a543b2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
ffdae68f247ef0c78da5bf053dd68b7b4e9aeeec8135bb75ae123a60fb25671b	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
ffe44295c63a13498687134a9e6ee5c57e08d84bfffa0d810821c1181b8873e1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
ffeb2a74486809e237cd05be056ca2e33b2141172e99d34753cec41fb661c2b3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
fff3573d9131d15da6a46c1ca8f03b5d37e4f6b804171ea27278a18a2d8ca0c7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

12306 rows × 46339 columns

❗ *The code snipplet below has to be run elsewhere** instead of Google Colab since it doesn't have enough RAM to complete the computation.*

In [8]:
```
from scipy.sparse.linalg import svds as svd
# Use SVD to retrieve the recommendation predication matrix
customers_ids = df.index.values.tolist()

# users_articles_mean is the average value of (sum of unique item purchased over total numbers of articles)
users_articles_mean = df.sum(axis=1) / len(df.columns)
users_articles_mean_np = users_articles_mean.to_numpy()

# df_subtract_mean records each entry after the substraction of average value from users_articles_mean_np
df_subtract_mean = df.to_numpy() - users_articles_mean_np.reshape(-1, 1)

# compute the largest or smallest k singular values and corresponding singular vectors of a sparse matrix df_subtract_mean
# left_singular_vec_U represents the relationship between users and latent factors
# singular_values_sigma describes the strength of each latent factor
# right_singular_vec_Vt indicates the similarity between items and latent factors
```

```
# Latent factors are the characteristics of the items
left_singular_vec_U, singular_values_sigma, right_singular_vec_Vt = scipy.sparse.linalg.svds(df_subtract_mean)

recommendation_mat = pd.DataFrame(np.dot(np.dot(left_singular_vec_U, np.diag(singular_values_sigma)), right_singular_vec_Vt) +
                        users_articles_mean_np.reshape(-1, 1),
                        columns = df.columns)
recommendation_mat
```

Out[8]:

article_id	108775015	108775044	108775051	110065001	110065002	110065011	111565001	111586001	111593001	111609001	...	939160001	939160002	941
0	0.000291	0.000313	0.000058	0.000096	0.000060	0.000066	0.000175	0.000458	0.000307	0.000068	...	0.000052	0.000052	0
1	0.000368	0.000333	0.000013	0.000066	0.000028	0.000046	0.000148	0.000489	0.000354	0.000055	...	0.000003	0.000003	0
2	0.006977	0.006180	0.000067	0.001141	0.000429	0.000847	0.002468	0.008977	0.006811	0.000901	...	-0.000148	-0.000148	-0
3	0.017076	0.012432	0.001221	0.003735	0.002421	0.003849	0.005195	0.016200	0.014937	0.004461	...	0.000644	0.000644	0
4	0.001645	0.002275	0.000432	0.000757	0.000530	0.000873	0.000879	0.001488	0.001533	0.000841	...	0.000372	0.000372	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
12301	0.004579	0.003611	0.000054	0.000734	0.000326	0.000620	0.001442	0.005252	0.004192	0.000752	...	-0.000091	-0.000091	-0
12302	-0.000075	-0.000057	0.000025	0.000012	0.000020	0.000016	-0.000008	-0.000101	-0.000066	0.000013	...	0.000028	0.000028	0
12303	0.007741	0.006772	0.000044	0.001115	0.000370	0.000642	0.002897	0.010700	0.007567	0.000757	...	-0.000171	-0.000171	-0
12304	0.004144	0.002246	0.000028	0.000347	0.000343	0.000481	0.000800	0.003800	0.003157	0.000739	...	-0.000096	-0.000096	-0
12305	0.040422	0.008205	0.004172	0.003021	0.007475	0.005262	0.005956	0.033245	0.028120	0.008262	...	0.003240	0.003240	0

12306 rows × 46339 columns

Recommendation System

In [9]:
```
df['index'] = df.index
df.index = np.arange(1, len(df) + 1)
```

In [10]:
```
def recommend(userID, num_of_predictions):
  # lookup user's id
  userID_alt = df.loc[df['index'] == userID].index[0]
  predicted_articles = recommendation_mat.iloc[userID_alt].sort_values(ascending=False)[:num_of_predictions].to_frame()
```

```
  predicted_articles['article_id'] = predicted_articles.index
  predicted_articles.index = np.arange(0, len(predicted_articles))
  results = []
  # define a dict {customer_id: def num_tran==2? 1: num_tran*0.2}
  for i in range(0, num_of_predictions):
    temp = m1_articles.loc[m1_articles['article_id']==predicted_articles.iloc[i]['article_id']].to_dict()
    results.append(temp)
  for article in results:
    for key in article.keys():
      article[key] = article[key][list(article[key].keys())[0]]
  return pd.DataFrame().from_dict(results)
```

In [11]:
```
recommend('000362878a3904e1fe4927bbfcdb10c64a9d85b12a593a7d9c965efcc9187cbf', 10)
```

```
C:\Users\ltxom\AppData\Local\Temp\ipykernel_22628\692544571.py:4: FutureWarning: The behavior of `series[i:j]` with an integer-dtype index is deprecated. In a future version, this will be treated as *label-based* indexing, consistent with e.g. `series[i]` lookups. To retain the old behavior, use `series.iloc[i:j]`. To get the future behavior, use `series.loc[i:j]`.
  predicted_articles = recommendation_mat.iloc[userID_alt].sort_values(ascending=False)[:num_of_predictions].to_frame()
```

Out[11]:

	article_id	product_code	prod_name	product_type_no	product_type_name	product_group_name	graphical_appearance_no	graphical_appearance_name	colour_gr
0	372860001	372860	7p Basic Shaftless	302	Socks	Socks & Tights	1010016	Solid	
1	610776002	610776	Tilly (1)	255	T-shirt	Garment Upper body	1010016	Solid	
2	610776001	610776	Tilly (1)	255	T-shirt	Garment Upper body	1010016	Solid	
3	720125001	720125	SUPREME RW tights	273	Leggings/Tights	Garment Lower body	1010016	Solid	
4	372860002	372860	7p Basic Shaftless	302	Socks	Socks & Tights	1010016	Solid	
5	351484002	351484	Lazer Razer Brief	59	Swimwear bottom	Swimwear	1010016	Solid	
6	464297007	464297	Greta Thong Mynta Low 3p	286	Underwear bottom	Underwear	1010014	Placement print	
7	579541001	579541	Calista cardigan.	245	Cardigan	Garment Upper body	1010016	Solid	

	article_id	product_code	prod_name	product_type_no	product_type_name	product_group_name	graphical_appearance_no	graphical_appearance_name	colour_gr
8	590928001	590928	New Girl Push Top	298	Bikini top	Swimwear	1010026	Other structure	
9	759871002	759871	Tilda tank	253	Vest top	Garment Upper body	1010016	Solid	

10 rows × 25 columns

Evaluation

In [12]:
```
from collections import OrderedDict
def get_prediction_mat(userID):
  # lookup user's id
  userID_alt = df.loc[df['index'] == userID].index[0]
  try:
    d = recommendation_mat.iloc[userID_alt].sort_values(ascending=False).to_frame().to_dict()[userID_alt]
  except IndexError:
    return None
  return d
```

In [13]:
```
# build customer to article dictionary from testing dataset for evaluation
counter = 0
customer_to_article_test = {}
for id in filtered_customer_ids:
  if counter % 1000 == 0:
    print('progress:', counter / len(filtered_customer_ids), '%')
  counter += 1
  actual_y = m1_test.loc[m1_test['customer_id']==id]['article_id'].to_list()
  customer_to_article_test[id] = actual_y
```

```
progress: 0.0 %
progress: 0.08126117341134406 %
progress: 0.16252234682268812 %
progress: 0.24378352023403219 %
progress: 0.32504469364537625 %
progress: 0.4063058670567203 %
progress: 0.48756704046806437 %
progress: 0.5688282138794084 %
progress: 0.6500893872907525 %
progress: 0.7313505607020966 %
progress: 0.8126117341134406 %
progress: 0.8938729075247847 %
progress: 0.9751340809361287 %
```

In [14]:

```python
# prediction: build recommendation for each customer
customer_to_prediction_dict = {}
counter = 0
for id in filtered_customer_ids:
  if counter % 1000 == 0:
    print('progress:', counter / len(filtered_customer_ids), '%')
  counter += 1
  t = get_prediction_mat(id)
  if t is not None:
    customer_to_prediction_dict[id] = t
```

```
progress: 0.0 %
progress: 0.08126117341134406 %
progress: 0.16252234682268812 %
progress: 0.24378352023403219 %
progress: 0.32504469364537625 %
progress: 0.4063058670567203 %
progress: 0.48756704046806437 %
progress: 0.5688282138794084 %
progress: 0.6500893872907525 %
progress: 0.7313505607020966 %
progress: 0.8126117341134406 %
progress: 0.8938729075247847 %
progress: 0.9751340809361287 %
```

In [15]:

```python
sum = 0
count = 0
counter = 0
for key in customer_to_prediction_dict.keys():
  if counter % 1000 == 0:
```

```
    print('progress:', counter / len(customer_to_prediction_dict), '%')
  counter += 1
  for score in customer_to_prediction_dict[key]:
    sum += customer_to_prediction_dict[key][score]
    count += 1

average_weight = sum / count
print(average_weight)
```

```
progress: 0.0 %
progress: 0.08126777732629012 %
progress: 0.16253555465258024 %
progress: 0.24380333197887039 %
progress: 0.3250711093051605 %
progress: 0.4063388866314506 %
progress: 0.48760666395774077 %
progress: 0.5688744412840309 %
progress: 0.650142218610321 %
progress: 0.7314099959366112 %
progress: 0.8126777732629012 %
progress: 0.8939455505891913 %
progress: 0.9752133279154815 %
0.0003737381245795775
```

In [17]:

```
sum = 0
count = 0
counter = 0
for key in customer_to_prediction_dict.keys():
  if counter % 1000 == 0:
    print('progress:', counter / len(customer_to_prediction_dict), '%')
  counter += 1
  top = 0
  for score in customer_to_prediction_dict[key]:
    if top > TEST_SIZE:
      break
    top += 1
    sum += customer_to_prediction_dict[key][score]
    count += 1

no_smoothing_weight = sum / count
print(no_smoothing_weight)
```

```
progress: 0.0 %
progress: 0.08126777732629012 %
progress: 0.16253555465258024 %
progress: 0.24380333197887039 %
progress: 0.3250711093051605 %
progress: 0.4063388866314506 %
progress: 0.48760666395774077 %
progress: 0.5688744412840309 %
progress: 0.650142218610321 %
progress: 0.7314099959366112 %
progress: 0.8126777732629012 %
progress: 0.8939455505891913 %
progress: 0.9752133279154815 %
0.015697326044706163
```

In [18]:
```python
# generate prediction accuracy score
num_of_true_items = 0
num_of_total_items = 0
for id in filtered_customer_ids:
  for article in customer_to_article_test[id]:
    if id in customer_to_prediction_dict.keys() and article in customer_to_prediction_dict[id].keys():
      num_of_total_items += 1
      if customer_to_prediction_dict[id][article] > no_smoothing_weight:
        num_of_true_items += 1
print('accuracy score (without smoothing)', num_of_true_items/num_of_total_items)
```

```
accuracy score (without smoothing) 0.009652476258022107
```

In [19]:
```python
# generate prediction accuracy score
num_of_true_items = 0
num_of_total_items = 0
for id in filtered_customer_ids:
  for article in customer_to_article_test[id]:
    if id in customer_to_prediction_dict.keys() and article in customer_to_prediction_dict[id].keys():
      num_of_total_items += 1
      if customer_to_prediction_dict[id][article] > average_weight:
        num_of_true_items += 1
print('accuracy score (with smoothing factor weight > average weight)', num_of_true_items/num_of_total_items)
```

```
accuracy score (with smoothing factor weight > average weight) 0.3304329775640471
```

In [20]:
```python
# generate prediction accuracy score
num_of_true_items = 0
num_of_total_items = 0
for id in filtered_customer_ids:
```

```
  for article in customer_to_article_test[id]:
    if id in customer_to_prediction_dict.keys() and article in customer_to_prediction_dict[id].keys():
      num_of_total_items += 1
      if customer_to_prediction_dict[id][article] > 0:
        num_of_true_items += 1
print('accuracy score (with smoothing factor weight > 0)', num_of_true_items/num_of_total_items)
```

```
accuracy score (with smoothing factor weight > 0) 0.8723209188880624
```