Recommendation Systems in Sales

In [ ]:
```
# There are user sessions data available. Each session contains the list of viewed goods (id) and the list of purchased goods
# The task is creating recommendation system which proposes to user k-goods based on this data
```

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

In [2]:
```
with open('Data/coursera_sessions_train.txt', 'r') as f:
    sess_train = f.read().splitlines()
with open('Data/coursera_sessions_test.txt', 'r') as f:
    sess_test = f.read().splitlines()
```

In [5]:
```
# the data strusture: each row - the list of viewed goods (id) by user for one session and purchased list of goods after ";"
sess_train[8:12]
```

Out[5]:
```
['71,72,73,74;',
 '76,77,78;',
 '84,85,86,87,88,89,84,90,91,92,93,86;86',
 '114,77,115,116,117,118,119,120,121,120,122,123,124;']
```

In [ ]:
```
# The project is builded by the following way

# 1. Create the frequency matrices of viewed and purchased goods
# (good - # of viewes; good - # of purchases), then sort its out by frequency descending.
# Thus, we get the popularity matrix of views (PMV) and the popularity matrix of purchasing (PMP)

# 2. For the current session of views [id_1,id_2,...,id_n] create recommendations on k-doods from that list
# (k<=n), which are located in the top of PMV

# 3. For the current session of views [id_1,id_2,...,id_n] create recommendations on k-doods from that list
# (k<=n), which are located in the top of PMP

# 4. - 6. Check all the above on testing data set (using of course popularity matrices of the train data set)
```

In [ ]:
```
# Finally, the metrics for recommendations by purchase are slightly better on the test dataset.
# Wherein, the metrics on test dataset are very close to metrics on train dataset for the recommendations by views.
```

1. On the taining: creating the frequences of id in viewed and in purchased goods

In [ ]:
```
# Note: each id can be multiple viewed, all that views should be considered
```

In [3]:
```
# separate the matrix of sessions on viewed id's and purchased id's
# for the train dataset
sess_train_lp = []
for sess in sess_train:
    look_items, pur_items = sess.split(';')
    # apply function int() to all id's within the session
    look_items = list(map(int, look_items.split(',')))
    if len(pur_items) > 0:
        pur_items = list(map(int, pur_items.split(',')))
    else:
        pur_items = []
    sess_train_lp.append([look_items, pur_items])

# for the test dataset
sess_test_lp = []
for sess in sess_test:
    look_items, pur_items = sess.split(';')
    look_items = list(map(int, look_items.split(',')))
    if len(pur_items) > 0:
        pur_items = list(map(int, pur_items.split(',')))
    else:
        pur_items = []
    sess_test_lp.append([look_items, pur_items])
```

In [10]:
```
sess_train_lp[8:12]
```

Out[10]:
```
[[[71, 72, 73, 74], []],
 [[76, 77, 78], []],
 [[84, 85, 86, 87, 88, 89, 84, 90, 91, 92, 93, 86], [86]],
 [[114, 77, 115, 116, 117, 118, 119, 120, 121, 120, 122, 123, 124], []]]
```

In [4]:
```
# FOR VIEWED

# first put all the viewed id's in the list
sess_train_lid = []
for i in range(len(sess_train_lp)):
    for id in sess_train_lp[i][0]:
        sess_train_lid.append(id)

# then the array of unique id's with a counter
sess_train_l_cnt = np.transpose(np.unique(sess_train_lid, return_counts=True))
```

In [22]:
```
# array structure: i-th element is 2-dim raw [id, how many views at all]
sess_train_l_cnt
```

Out[22]:
```
array([[     0,      6],
       [     1,      6],
       [     2,      9],
       ...,
       [102804,      1],
       [102805,      1],
       [102806,      1]], dtype=int64)
```

In [5]:
```
# FOR PURCHASED

# first put all the purchased id's in the list
sess_train_pid = []
for i in range(len(sess_train_lp)):
    for id in sess_train_lp[i][1]:
        sess_train_pid.append(id)

# then the array of unique id's with a counter
sess_train_p_cnt = np.transpose(np.unique(sess_train_pid, return_counts=True))
```

In [24]:
```
sess_train_p_cnt
```

Out[24]:
```
array([[     5,      1],
       [     6,      2],
       [     7,      2],
       ...,
       [102417,      1],
       [102462,      1],
       [102646,      1]], dtype=int64)
```

In [6]:
```
# Sorting the created above arrays by counter (descending)
# sess_train_l_cnt[:,1].argsort() - list of raw indexes of our 2-dim matrix above by acsending counter
# sess_train_l_cnt[sess_train_l_cnt[:,1].argsort()] - the matrix sorted by acsending counter
# list[<start>:<stop>:<step>], [::-1] - the inverded list from end to begin
sess_train_l_cnt = sess_train_l_cnt[sess_train_l_cnt[:,1].argsort()][::-1]
sess_train_p_cnt = sess_train_p_cnt[sess_train_p_cnt[:,1].argsort()][::-1]
```

In [40]:
```
# the final matrix of purchased goods: good id - the number of purches
sess_train_p_cnt
```

Out[40]:
```
array([[  158,    14],
       [  204,    12],
       [   73,    11],
       ...,
       [38189,     1],
       [38177,     1],
       [    5,     1]], dtype=int64)
```

2. Recommendation alhorithm - sorting of the viewed id by popularity (frequency of viewed goods)

In [7]:
```
# metrics function 'Precision@k' and 'Recall@k' for k-recommendations
# 'reccomendations' - the list of 'k' recommended goods for the current session, where the list of purchases 'session'
def prec_rec_metrics(session, reccomendations, k):
    purchase = 0
    for ind in reccomendations:
        if ind in session:
            purchase += 1
    precision = purchase / k
    recall = purchase / len(session)
    return(precision, recall)
```

In [8]:
```
# list of purchased and viewed goods
sess_train_p = [row[1] for row in sess_train_lp]
sess_train_l = [row[0] for row in sess_train_lp]
```

In [9]:
```
%%time
# Calculate metrics 'Precision@k' and 'Recall@k' on the train dataset for k=1 and k=5
prec_at_1_tr_l, rec_at_1_tr_l = [], []
prec_at_5_tr_l, rec_at_5_tr_l = [], []
k1, k5 = 1, 5
for i, sess_p in enumerate(sess_train_p):
    # don't process the sessions without purchases
    if sess_p == []: continue

    # viewed id's for session i
    sess_l = sess_train_l[i]

    l_ind_sess = []
    for j in range(len(sess_l)):
        # select the index (!) of the counter-array of the meeted viewed id,
        # where the element = the current id of the session.
        l_ind_sess.append(np.where(sess_train_l_cnt[:,0] == sess_l[j])[0][0])
    # the list of indices of the matrix-counter-of-the-purchased-goods for the purchased goods of the current session
    l_ind_sess_sorted = np.unique(l_ind_sess)

    # k1 recommendations
    num_of_recs_k1 = min(k1, len(sess_l))
    if num_of_recs_k1 == 0: continue
    # take the first Min(k1,# of id's in the current session) # of id's from the current session,
    # according to the sorting by view frequency on the whole train dataset
    recs_k1 = sess_train_l_cnt[l_ind_sess_sorted[:num_of_recs_k1],0]

    # k1-metrics calcs
    prec_1, rec_1 = prec_rec_metrics(sess_p, recs_k1, k1)
    prec_at_1_tr_l.append(prec_1)
    rec_at_1_tr_l.append(rec_1)

    # k5 recommendations
    num_of_recs_k5 = min(k5, len(sess_l))
    if num_of_recs_k5 == 0: continue
    recs_k5 = sess_train_l_cnt[l_ind_sess_sorted[:num_of_recs_k5],0]

    # k5-metrics calcs
    prec_5, rec_5 = prec_rec_metrics(sess_p, recs_k5, k5)
    prec_at_5_tr_l.append(prec_5)
    rec_at_5_tr_l.append(rec_5)
```

```
Wall time: 8.23 s
```

In [10]:
```
avg_prec_at_1_tr_l = np.mean(prec_at_1_tr_l)
avg_rec_at_1_tr_l = np.mean(rec_at_1_tr_l)
avg_prec_at_5_tr_l = np.mean(prec_at_5_tr_l)
avg_rec_at_5_tr_l = np.mean(rec_at_5_tr_l)
```

In [13]:
```
r1 = round(avg_rec_at_1_tr_l, 2)
p1 = round(avg_prec_at_1_tr_l, 2)
r5 = round(avg_rec_at_5_tr_l, 2)
p5 = round(avg_prec_at_5_tr_l, 2)
```

In [24]:
```
# DataFrame 'metrics' of the results of this project
metrics = pd.DataFrame({'precision':p1, 'recall': r1},index={'Train_look_k=1'})
metrics.loc['Train_look_k=5'] = [p5,r5]
```

3. Recommendation alhorithm - sorting of the purchases id by popularity (frequency of purchased goods)

In [25]:
```
%%time
# Precision@k' and 'Recall@k' on train for k=1 and k=5
prec_at_1_tr_p, rec_at_1_tr_p = [], []
prec_at_5_tr_p, rec_at_5_tr_p = [], []
k1, k5 = 1, 5

for i, sess_p in enumerate(sess_train_p):
    # no processing without purchases
    if sess_p == []: continue

    # viewed id's for session i
    sess_l = sess_train_l[i]

    l_ind_sess = []
    for j in range(len(sess_l)):
        # select the index of counter-array of the meeted viewed id,
        # where the element = the current id of the session
        # here the first [0]-th element selects the first raw in the founded 1-raw element,
        # the second [0] selects the first raw element, which is an index
        if sess_l[j] not in sess_train_p_cnt[:,0]: continue
        l_ind_sess.append(np.where(sess_train_p_cnt[:,0] == sess_l[j])[0][0])
    l_ind_sess_sorted = np.unique(l_ind_sess)

    # k1 recommendations
    num_of_recs_k1 = min(k1, len(sess_l), len(l_ind_sess_sorted))
    if num_of_recs_k1 == 0: continue
    #
    #
    recs_k1 = sess_train_p_cnt[l_ind_sess_sorted[:num_of_recs_k1],0]

    # k1-metrics
    prec_1, rec_1 = prec_rec_metrics(sess_p, recs_k1, k1)
    prec_at_1_tr_p.append(prec_1)
    rec_at_1_tr_p.append(rec_1)

    # k5 recommendations
    num_of_recs_k5 = min(k5, len(sess_l), len(l_ind_sess_sorted))
    if num_of_recs_k5 == 0: continue
    recs_k5 = sess_train_p_cnt[l_ind_sess_sorted[:num_of_recs_k5],0]

    # k5-metrics
    prec_5, rec_5 = prec_rec_metrics(sess_p, recs_k5, k5)
    prec_at_5_tr_p.append(prec_5)
    rec_at_5_tr_p.append(rec_5)
```

```
Wall time: 1.74 s
```

In [26]:
```
avg_prec_at_1_tr_p = np.mean(prec_at_1_tr_p)
avg_rec_at_1_tr_p = np.mean(rec_at_1_tr_p)
avg_prec_at_5_tr_p = np.mean(prec_at_5_tr_p)
avg_rec_at_5_tr_p = np.mean(rec_at_5_tr_p)

r1 = round(avg_rec_at_1_tr_p, 2)
p1 = round(avg_prec_at_1_tr_p, 2)
r5 = round(avg_rec_at_5_tr_p, 2)
p5 = round(avg_prec_at_5_tr_p, 2)
```

In [27]:
```
metrics.loc['Train_purch_k=1'] = [p1,r1]
metrics.loc['Train_purch_k=5'] = [p5,r5]
```

4. Build the frequency of id in viewed and purchased goods on the test dataset

In [53]:
```
# Make the array of viewed goods for all the train sessions
sess_test_l = [row[0] for row in sess_test_lp]
sess_test_l_np = []
for sess in sess_test_l:
    for idd in sess:
        sess_test_l_np.append(idd)
sess_test_l_np = np.array(sess_test_l_np)

# Make the array of purchased goods for all the train sessions
sess_test_p = [row[1] for row in sess_test_lp]
sess_test_p_np = []
for sess in sess_test_p:
    for idd in sess:
        sess_test_p_np.append(idd)
sess_test_p_np = np.array(sess_test_p_np)
```

5. Recommendation alhorithm on the test dataset - sorting of the viewed id by popularity (frequency of viewed goods)

In [57]:
```
%%time
#
prec_at_1_tst_l, rec_at_1_tst_l = [], []
prec_at_5_tst_l, rec_at_5_tst_l = [], []
k1, k5 = 1, 5

for i, sess_p in enumerate(sess_test_p):
    #
    if sess_p == []: continue

    #
    sess_l = sess_test_l[i]

    # select the index of counter-array of the meeted viewed id in PMV
    l_ind_sess = []
    new_ids = []
    for j in range(len(sess_l)):
        if sess_l[j] not in sess_train_l_cnt[:,0]:
            new_ids.append(sess_l[j])
            continue
        l_ind_sess.append(np.where(sess_train_l_cnt[:,0] == sess_l[j])[0][0])
    l_ind_sess_sorted = np.unique(l_ind_sess)

    # k1
    num_of_recs_k1 = min(k1, len(sess_l))
    if num_of_recs_k1 == 0: continue
    # add this condition as all viewed goods in the session can be not in the train matrix
    if l_ind_sess != []:
        recs_k1 = sess_train_l_cnt[l_ind_sess_sorted[:num_of_recs_k1],0]
    else:
        recs_k1 = []
    # here we join the sorted by 'decsending of the priority of PMV'
    # recommended goods and new goods which are not in PMV
    recs_k1 = np.concatenate((np.array(recs_k1, dtype='int64'), np.unique(np.array(new_ids, dtype='int64'))))[:num_of_recs_k1]

    # k1 metrics
    prec_1, rec_1 = prec_rec_metrics(sess_p, recs_k1, k1)
    prec_at_1_tst_l.append(prec_1)
    rec_at_1_tst_l.append(rec_1)

    # k5 recomm
    num_of_recs_k5 = min(k5, len(sess_l))
    if num_of_recs_k5 == 0: continue
    if l_ind_sess != []:
        recs_k5 = sess_train_l_cnt[l_ind_sess_sorted[:num_of_recs_k5],0]
    else:
        recs_k5 = []
    recs_k5 = np.concatenate((np.array(recs_k5, dtype='int64'), np.unique(np.array(new_ids, dtype='int64'))))[:num_of_recs_k5]

    # k5 metrics
    prec_5, rec_5 = prec_rec_metrics(sess_p, recs_k5, k5)
    prec_at_5_tst_l.append(prec_5)
    rec_at_5_tst_l.append(rec_5)
```

```
Wall time: 12.9 s
```

In [59]:
```
avg_prec_at_1_tst_l = np.mean(prec_at_1_tst_l)
avg_rec_at_1_tst_l = np.mean(rec_at_1_tst_l)
avg_prec_at_5_tst_l = np.mean(prec_at_5_tst_l)
avg_rec_at_5_tst_l = np.mean(rec_at_5_tst_l)

r1 = round(avg_rec_at_1_tst_l, 2)
p1 = round(avg_prec_at_1_tst_l, 2)
r5 = round(avg_rec_at_5_tst_l, 2)
p5 = round(avg_prec_at_5_tst_l, 2)
```

In [60]:
```
metrics.loc['Test_look_k=1'] = [p1,r1]
metrics.loc['Test_look_k=5'] = [p5,r5]
```

6. Recommendation alhorithm on the test dataset - sorting of the purchased id by popularity (frequency of purchased goods)

In [84]:
```
prec_at_1_tst_p, rec_at_1_tst_p = [], []
prec_at_5_tst_p, rec_at_5_tst_p = [], []
k1, k5 = 1, 5
for i, sess_p in enumerate(sess_test_p):

    if sess_p == []: continue


    sess_l = sess_test_l[i]


    l_ind_sess = []
    new_ids = []
    for j in range(len(sess_l)):
        if sess_l[j] not in sess_train_p_cnt[:,0]:
            new_ids.append(sess_l[j])
            continue
        l_ind_sess.append(np.where(sess_train_p_cnt[:,0] == sess_l[j])[0][0])
    l_ind_sess_sorted = np.unique(l_ind_sess)
    # k1 recs
    num_of_recs_k1 = min(k1, len(sess_l))
    if num_of_recs_k1 == 0: continue
    if l_ind_sess != []:
        recs_k1 = sess_train_p_cnt[l_ind_sess_sorted[:num_of_recs_k1],0]
    else:
        recs_k1 = []
    #
    recs_k1 = np.concatenate((np.array(recs_k1, dtype='int64'), np.unique(np.array(new_ids, dtype='int64'))))[:num_of_recs_k1]

    # k1 metrics
    prec_1, rec_1 = prec_rec_metrics(sess_p, recs_k1, k1)
    prec_at_1_tst_p.append(prec_1)
    rec_at_1_tst_p.append(rec_1)

    # k5 recommend
    num_of_recs_k5 = min(k5, len(sess_l))
    if num_of_recs_k5 == 0: continue
    if l_ind_sess != []:
        recs_k5 = sess_train_p_cnt[l_ind_sess_sorted[:num_of_recs_k5],0]
    else:
        recs_k5 = []
    recs_k5 = np.concatenate((np.array(recs_k5, dtype='int64'), np.unique(np.array(new_ids, dtype='int64'))))[:num_of_recs_k5]

    # k5 metrics
    prec_5, rec_5 = prec_rec_metrics(sess_p, recs_k5, k5)
    prec_at_5_tst_p.append(prec_5)
    rec_at_5_tst_p.append(rec_5)
```

In [85]:
```
avg_prec_at_1_tst_p = np.mean(prec_at_1_tst_p)
avg_rec_at_1_tst_p = np.mean(rec_at_1_tst_p)
avg_prec_at_5_tst_p = np.mean(prec_at_5_tst_p)
avg_rec_at_5_tst_p = np.mean(rec_at_5_tst_p)

r1 = round(avg_rec_at_1_tst_p, 2)
p1 = round(avg_prec_at_1_tst_p, 2)
r5 = round(avg_rec_at_5_tst_p, 2)
p5 = round(avg_prec_at_5_tst_p, 2)
```

```
Answer 4: 0.42 0.49 0.80 0.20
```

In [86]:
```
metrics.loc['Test_purch_k=1'] = [p1,r1]
metrics.loc['Test_purch_k=5'] = [p5,r5]
```

7. Analysis of the results

In [87]:
```
metrics
```

Out[87]:

	precision	recall
Train_look_k=1	0.51	0.44
Train_look_k=5	0.21	0.83
Train_purch_k=1	0.79	0.68
Train_purch_k=5	0.25	0.93
Test_look_k=1	0.48	0.42
Test_look_k=5	0.2	0.8
Test_purch_k=1	0.49	0.42
Test_purch_k=5	0.20	0.80

In [93]:
```
metrics.loc[['Train_look_k=1','Test_purch_k=1']]
```

Out[93]:

	precision	recall
Train_look_k=1	0.51	0.44
Test_purch_k=1	0.49	0.42

In [94]:
```
metrics.loc[['Train_look_k=5','Test_purch_k=5']]
```

Out[94]:

	precision	recall
Train_look_k=5	0.21	0.83
Test_purch_k=5	0.20	0.80

In [95]:
```
metrics.loc[['Train_look_k=1','Test_look_k=1']]
```

Out[95]:

	precision	recall
Train_look_k=1	0.51	0.44
Test_look_k=1	0.48	0.42

In [96]:
```
metrics.loc[['Train_look_k=5','Test_look_k=5']]
```

Out[96]:

	precision	recall
Train_look_k=5	0.21	0.83
Test_look_k=5	0.2	0.8

In [97]:
```
metrics.loc[['Train_purch_k=1','Test_purch_k=1']]
```

Out[97]:

	precision	recall
Train_purch_k=1	0.79	0.68
Test_purch_k=1	0.49	0.42

In [98]:
```
metrics.loc[['Train_purch_k=5','Test_purch_k=5']]
```

Out[98]:

	precision	recall
Train_purch_k=5	0.25	0.93
Test_purch_k=5	0.20	0.80

In [ ]: