In [1]:

```
import pandas as pd
import implicit
import os
import numpy as np
from scipy import sparse
```

```
Решается задача товарных рекомендаций.

- `purchases_train.csv` - история покупок в розничном магазине (с 21 октября 2003 года по 12 марта 2004 года)

- `purchases_test.csv` - покупки за следующую неделю (с 13 по 19 марта 2004 года). В этой выборке для каждого пользователя исключены товары, которые он уже покупал за период обучающей выборки

- `customers.csv` - пол клиентов

В решении ниже
- обучается модель матричного разложения AlternateLeastSquares
- сравнивается с тестовыми данными
- измеряется ее качество по метрике map@10
```

In [2]:

```
purchases_train = pd.read_csv('purchases_train.csv', dtype = {"customer_id": "int32", "prod
purchases_train.head()
```

Out[2]:

| | customer_id | product_id | datetime |
|---|---|---|---|
| **0** | 8698595 | 12530 | 2004-03-10 22:18:43.497459200 |
| **1** | 13271885 | 7541 | 2004-03-06 02:24:43.209763200 |
| **2** | 16852746 | 13134 | 2004-03-10 01:03:09.598614400 |
| **3** | 16852746 | 6572 | 2004-03-04 16:45:16.522566400 |
| **4** | 14619070 | 4659 | 2004-03-12 13:29:35.011481600 |

In [3]:

```
purchases_train['product_id'].describe()
```

Out[3]:

```
count    351686.000000
mean       6705.034394
std        3979.766820
min          15.000000
25%        3121.000000
50%        6189.000000
75%       10228.000000
max       13830.000000
Name: product_id, dtype: float64
```

Обучаем модель AlternateLeastSquares

In [4]:

```
def get_csr_matrix(customer_id, product_id):
    row = np.array(customer_id)
    col = np.array(product_id)
    data = np.array(np.ones(customer_id.size, dtype=np.float32))
    return sparse.coo_matrix((data, (row, col))).tocsr()
```

In [7]:

```
user_items = get_csr_matrix(purchases_train.customer_id, purchases_train.product_id)
item_users = user_items.T.tocsr()
```

In [8]:

```
model = implicit.als.AlternatingLeastSquares(factors=32, iterations=100)
```

```
WARNING:root:Intel MKL BLAS detected. Its highly recommend to set the enviro
nment variable 'export MKL_NUM_THREADS=1' to disable its internal multithrea
ding
```

In [9]:

```
np.random.seed(42)
model.fit(item_users=item_users)
```

```
100%                                                  100/100 [06:23<00:00, 3.84s/it]
```

purchases_test.csv содержит данные о покупках с 13 марта 2004 по 20 марта 2004 - то есть неделя следующая за обучающей выборкой

для каждого пользователя исключены те товары, которые он покупал в обучающей выборке

In [10]:

```
purchases_test = pd.read_csv('purchases_test.csv')
purchases_test.head()
```

Out[10]:

| | customer_id | product_id | datetime |
|---|---|---|---|
| **0** | 1021292 | 6197 | 2004-03-18 13:35:19.145152000 |
| **1** | 11379978 | 4659 | 2004-03-19 18:51:31.887936000 |
| **2** | 13271885 | 5659 | 2004-03-14 05:47:21.544166400 |
| **3** | 13271885 | 1015 | 2004-03-15 14:41:19.702089601 |
| **4** | 12315337 | 12072 | 2004-03-19 10:39:17.148105600 |

Измеряем качество рекомендаций с помощью метрики map@10

In [11]:

```
relevant = purchases_test.groupby('customer_id')['product_id'].apply(lambda s: s.values).re
relevant.rename(columns={'product_id': 'product_ids'}, inplace=True)
relevant.head()
```

Out[11]:

| | customer_id | product_ids |
|---|---|---|
| **0** | 107 | [5868] |
| **1** | 453 | [11854] |
| **2** | 1011 | [10609, 7110] |
| **3** | 1135 | [8994] |
| **4** | 2947 | [5868] |

In [12]:

```
recommendations = []
for user_id in relevant['customer_id']:
    recommendations.append([x[0] for x in model.recommend(userid=user_id, user_items=user_i
```

In [13]:

```
def apk(actual, predicted, k=10):
    """
    Computes the average precision at k.
    This function computes the average prescision at k between two lists of
    items.
    Parameters
    ----------
    actual : list
             A list of elements that are to be predicted (order doesn't matter)
    predicted : list
                A list of predicted elements (order does matter)
    k : int, optional
        The maximum number of predicted elements
    Returns
    -------
    score : double
            The average precision at k over the input lists
    """
    if len(predicted)>k:
        predicted = predicted[:k]

    score = 0.0
    num_hits = 0.0

    for i,p in enumerate(predicted):
        if p in actual and p not in predicted[:i]:
            num_hits += 1.0
            score += num_hits / (i+1.0)

    if len(actual) == 0:
        return 0.0

    return score / min(len(actual), k)

def mapk(actual, predicted, k=10):
    """
    Computes the mean average precision at k.
    This function computes the mean average prescision at k between two lists
    of lists of items.
    Parameters
    ----------
    actual : list
             A list of lists of elements that are to be predicted
             (order doesn't matter in the lists)
    predicted : list
                A list of lists of predicted elements
                (order matters in the lists)
    k : int, optional
        The maximum number of predicted elements
    Returns
    -------
    score : double
            The mean average precision at k over the input lists
    """
    return np.mean([apk(a,p,k) for a,p in zip(actual, predicted)])
```

In [15]:

```
print("Качество модели map10:", mapk(relevant['product_ids'], recommendations, k=10))
```

```
Качество модели map10: 0.15407707481592564
```

In [ ]:

```

```