Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Лабораторная работа №4
«Создание рекомендательной модели»

ИСПОЛНИТЕЛЬ:

Елизаров Олег Олегович
Группа ИУ5-21М

____

"__"__________2022 г.

Москва 2022

Целью работы является: изучение разработки рекомендательных моделей.

Задание:

1. Выбрать произвольный набор данных (датасет), предназначенный для построения рекомендательных моделей.
2. Опираясь на материалы лекции, сформировать рекомендации для одного пользователя (объекта) двумя произвольными способами.
3. Сравнить полученные рекомендации (если это возможно, то с применением метрик).

Для выполнения данной работы взят датасет с данными пользователей об их транзакциях. Чтобы грамотно предлагать в рекламных баннерах конкретному пользователю именно те услуги, которые его заинтересуют больше остальных.

In [10]:
```
import pandas as pd
import math
import numpy as np
import vertica_python
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix
from sklearn.metrics import accuracy_score, balanced_accuracy_score, precision_score, recall_score
from sklearn.cluster import KMeans, MiniBatchKMeans, DBSCAN
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

In [3]:
```
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', 60)
pd.set_option('display.width', 1000)
```

Переделка признаков с максимальными датами на бинарные признаки

n [18]:
```
def rule(x):
    if pd.isna(x):
        return 0
    else:
        return 1
```

n [19]:
```
columns = ['android_app', 'ios_app', 'site_app', 'category_1', 'category_2', 'category_3', 'category_4', 'category_5
```

n [20]:
```
for i in columns:
    from_vertica[i+'_bin'] = from_vertica.apply(lambda x: rule(x[i]), axis =  1)
    from_vertica = from_vertica.drop(i, 1)
from_vertica.head()
```

ut[20]:

	user_id	registration_date	success_payments	unsuccess_payments	priority_package	sms_package	qvc_cards	qvp_cards	android_app_bin	ios_app_bin	s
0	0	2020-01-01	29	1.0	NaN	NaN	NaN	NaN	1	0	
1	1	2020-01-01	6	NaN	NaN	NaN	1.0	NaN	1	0	
2	2	2020-01-01	20	1.0	NaN	NaN	NaN	NaN	1	0	
3	3	2020-01-01	3	NaN	NaN	NaN	NaN	NaN	1	0	
4	4	2020-01-01	1	128.0	NaN	NaN	NaN	NaN	1	0	

Избавление от нулевых значений

:1]:
```
u = from_vertica.select_dtypes(include=['datetime'])
from_vertica[u.columns] = u.fillna(0)
from_vertica
```

:1]:

	user_id	registration_date	success_payments	unsuccess_payments	priority_package	sms_package	qvc_cards	qvp_cards	android_app_bin	ios_app
0	0	2020-01-01	29	1.0	NaN	NaN	NaN	NaN	1	
1	1	2020-01-01	6	NaN	NaN	NaN	1.0	NaN	1	
2	2	2020-01-01	20	1.0	NaN	NaN	NaN	NaN	1	

```python
u = from_vertica.select_dtypes(exclude=['datetime'])
from_vertica[u.columns] = u.fillna(0)
from_vertica
```

	user_id	registration_date	success_payments	unsuccess_payments	priority_package	sms_package	qvc_cards	qvp_cards	android_app_bin	ios_app
0	0	2020-01-01	29	1.0	0.0	0.0	0.0	0.0	1	
1	1	2020-01-01	6	0.0	0.0	0.0	1.0	0.0	1	
2	2	2020-01-01	20	1.0	0.0	0.0	0.0	0.0	1	
3	3	2020-01-01	3	0.0	0.0	0.0	0.0	0.0	1	
4	4	2020-01-01	1	128.0	0.0	0.0	0.0	0.0	1	
...	...	...	...	...	...	...	...	...	...	
1865174	1865174	2020-03-13	2	0.0	0.0	0.0	0.0	0.0	0	
1865175	1865175	2020-03-13	4	0.0	0.0	0.0	0.0	0.0	0	
1865176	1865176	2020-03-13	9	0.0	0.0	0.0	0.0	0.0	1	
1865177	1865177	2020-03-13	1	0.0	0.0	0.0	0.0	0.0	0	
1865178	1865178	2020-03-13	2	0.0	0.0	0.0	0.0	0.0	0	

1865179 rows × 29 columns

```python
le = LabelEncoder()
from_vertica['registration_date'] = le.fit_transform(from_vertica['registration_date'])
```

```python
from_vertica['registration_date'].unique()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 34, 35, 36, 37, 12,
       13, 14, 38, 39, 40, 41, 15, 16, 42, 43, 44, 17, 18, 45, 46, 50, 47,
       48, 19, 20, 21, 22, 49, 23, 24, 51, 52, 25, 26, 27, 53, 54, 28, 29,
       55, 30, 31, 32, 56, 57, 58, 33, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 72, 71, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87])
```

```python
from_vertica.head(77)
```

	user_id	registration_date	success_payments	unsuccess_payments	priority_package	sms_package	qvc_cards	qvp_cards	android_app_bin	ios_app_bin
0	0	0.0	0.001584	0.000037	0.0	0.0	0.0	0.0	1	0
1	1	0.0	0.000283	0.000000	0.0	0.0	1.0	0.0	1	0
2	2	0.0	0.001075	0.000037	0.0	0.0	0.0	0.0	1	0
3	3	0.0	0.000113	0.000000	0.0	0.0	0.0	0.0	1	0

Масштабирование данных

```
sc1 = MinMaxScaler()
```

```
columnsForScaling = ['registration_date', 'success_payments', 'unsuccess_payments']
```

```
for i in columnsForScaling:
    from_vertica[i] = sc1.fit_transform(from_vertica[[i]])
```

```
from_vertica[columnsForScaling].describe()
```

	registration_date	success_payments	unsuccess_payments
count	1.865179e+06	1.865179e+06	1.865179e+06
mean	4.734473e-01	4.523057e-04	4.279049e-05
std	2.848241e-01	2.150745e-03	1.270634e-03
min	0.000000e+00	0.000000e+00	0.000000e+00
25%	2.298851e-01	0.000000e+00	0.000000e+00
50%	4.712644e-01	5.658669e-05	0.000000e+00
75%	7.126437e-01	2.829335e-04	0.000000e+00
max	1.000000e+00	1.000000e+00	1.000000e+00

Визуализация данных

```
columnsForVizualization = ['registration_date', 'success_payments', 'android_app_bin', 'ios_app_bin', 'sms_package',
```

```
sns.pairplot(from_vertica[columnsForVizualization])
```

```
from_vertica.corr()
```

	user_id	registration_date	success_payments	unsuccess_payments	priority_package	sms_package	qvc_cards	qvp_cards	android_app
user_id	1.000000	0.524667	-0.026066	-0.005667	0.000408	0.000021	-0.012046	-0.005269	-0.048
registration_date	0.524667	1.000000	-0.056443	-0.012909	-0.000304	-0.000952	-0.020361	-0.011219	-0.098
success_payments	-0.026066	-0.056443	1.000000	0.060624	0.018296	0.018909	0.086604	0.044887	0.077
unsuccess_payments	-0.005667	-0.012909	0.060624	1.000000	0.001367	0.001539	0.018931	0.004278	0.010
priority_package	0.000408	-0.000304	0.018296	0.001367	1.000000	0.967644	0.007485	0.247845	-0.002
sms_package	0.000021	-0.000952	0.018909	0.001539	0.967644	1.000000	0.008760	0.241145	-0.00
qvc_cards	-0.012046	-0.020361	0.086604	0.018931	0.007485	0.008760	1.000000	0.033050	-0.020
qvp_cards	-0.005269	-0.011219	0.044887	0.004278	0.247845	0.241145	0.033050	1.000000	0.029
android_app_bin	-0.048013	-0.098988	0.077130	0.010795	-0.002130	-0.001000	-0.020917	0.029411	1.000
ios_app_bin	0.010073	0.032750	0.007298	-0.001158	0.001570	0.001888	0.199368	-0.001700	-0.262
site_app_bin	-0.029673	-0.066284	0.057370	0.002870	0.021629	0.021842	0.133071	0.044472	-0.127
category_1_bin	-0.060390	-0.104810	0.124002	0.007469	0.002452	0.002969	0.201041	0.018394	0.069
category_2_bin	-0.024120	-0.042166	0.045080	0.007115	-0.002954	-0.003057	-0.039734	-0.007585	0.148
category_3_bin	-0.009307	-0.018301	0.024702	0.000633	0.002692	0.002820	0.009129	0.017951	0.024
category_4_bin	-0.002220	-0.005156	0.005192	0.000179	-0.000441	-0.000456	-0.001641	0.005090	0.01
category_5_bin	-0.001910	-0.002956	0.012293	0.000736	0.008649	0.008346	0.013387	0.002852	0.008
category_6_bin	-0.005721	-0.014964	0.088391	0.006142	0.002281	0.003563	0.124773	0.017263	0.27
category_7_bin	-0.002683	-0.004170	0.012309	0.000371	0.003463	0.003323	0.003875	0.011009	0.01
category_8_bin	-0.003570	-0.007164	0.012449	0.000681	0.002213	0.002067	-0.005087	0.000800	0.013

```
from_vertica.shape
```

(1865179, 29)

```
from_vertica.to_csv(r'data.csv', index = False, header=True)
```

```
from_vertica = pd.read_csv(r'data.csv', sep=",")
```

```
data.shape
```

(1865179, 29)

Модуль рекомендации

```
cols_x = ['user_id', 'registration_date', 'success_payments', 'unsuccess_payments', 'priority_package', 'sms_package
```

```
col_y = 'category_1_bin'
```

```
X7Cl = pd.read_csv(r'data.csv', sep=",")
X10Cl = pd.read_csv(r'data.csv', sep=",")
X15Cl = pd.read_csv(r'data.csv', sep=",")
YTrue = data[col_y]
```

Делю на 10 кластеров

```
Clusters7 = KMeans(n_clusters = 7).fit_predict(X7Cl)
```

```
X7Cl['Cluster'] = Clusters7
X7Cl['Cluster'].unique()
```

```
array([6, 3, 1, 4, 0, 5, 2])
```

Считаю средние значения по каждому кластеру

```
sumOfCluster7 = [0] * 7
count7 = [0] * 7

for index, row in X7Cl.iterrows():
    count7[row['Cluster'].astype(int)] += 1
    sumOfCluster7[row['Cluster'].astype(int)] += row['category_1_bin']


for i in range(len(sumOfCluster7)):
    print('sum7{} = {}'.format(i,sumOfCluster7[i]))
for i in range(len(count7)):
    print('count7{} = {}'.format(i,count7[i]))
```

```
sum70 = 58548.0
sum71 = 64103.0
sum72 = 62486.0
sum73 = 70276.0
sum74 = 61454.0
sum75 = 55743.0
sum76 = 81773.0
count70 = 267019
count71 = 266798
count72 = 266051
count73 = 266091
```

```
mean7 = [0] * 7
for i in range(len(sumOfCluster7)):
    mean7[i] = sumOfCluster7[i]/count7[i]

for i in range(len(mean7)):
    print('mean7{} = {}'.format(i,mean7[i]))
```

```
mean70 = 0.21926529572801937
mean71 = 0.24026791805036019
mean72 = 0.2348647439776584
mean73 = 0.26410513696442195
mean74 = 0.23000714865841015
mean75 = 0.20917247357341467
mean76 = 0.3079451992890067
```

```
best7 = []
for i in range(len(mean7)):
    if mean7[i] >= 0.25:
        best7.append(i)
best7
```

```
[3, 6]
```

Проставляю предсказанное значение

```
def rule2(x, best):
    if x in best:
        return 1
    else:
        return 0
```

```
X7Cl['YPred'] = X7Cl.apply(lambda x: rule2(x['Cluster'], best7), axis =  1)
```

Проверка качества предсказания

```
accuracy_score(YTrue, X7Cl['YPred'])
```

```
0.6343943396317459
```

```
confusion_matrix(YTrue, X7Cl['YPred'], labels=[0, 1])
```

```
array([[1031210,  379586],
       [ 302334,  152049]])
```

```
precision_score(YTrue, X7Cl['YPred']), recall_score(YTrue, X7Cl['YPred'])
```

```
(0.2860026145757898, 0.3346273958312701)
```

```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, pos_label, average):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

Коллаборативная фильтрация

```
XColFil = pd.read_csv(r'data.csv', sep=",")
YTrue = data[col_y]
```

```
def distCosine (usrA, usrB):
    def dotProduct (usrA, usrB):
        d = 0.0
        for dim in range(1, 29):
            d += usrA[dim]*usrB[dim]
        return d
    return dotProduct(usrA,usrB)/math.sqrt(dotProduct(usrA,usrA))/math.sqrt(dotProduct(usrB,usrB))
```

```
short = XColFil.head(2500)
```

```
mas = []

for index, row in short.iterrows():
    print(index)
    mas.append([])
    for index2, row2 in short.iterrows():
        mas[index].append(distCosine(row, row2))
```

```
5]: maspd = pd.DataFrame(mas)
```

```
6]: maspd.to_csv(r'cosinus.csv', index = False, header=True)
```

```
7]: best = []
    for i in mas:
        arr = np.array(i)
        np.nan_to_num(arr, 0)
        best.append(np.argpartition(arr, -5)[-5:])
```

```
8]: best
```

```
    array([ 299, 1314,  141,   16,  214]),
    array([ 165,  391,  787,  996, 1154]),
    array([ 191, 1294,  374,  410,   18]),
    array([1234,  751,   19,  869,  278]),
    array([ 377, 1515,   84,  105,   20]),
    array([ 720,   21,  236, 1091,  762]),
    array([983,  12, 529,  22, 959]),
```

```
bestPD = pd.DataFrame(best)
```

```
bestPD.to_csv(r'best.csv', index = False, header=True)
```

```
YPredCol = []
for i in range(0, 2500):
    y = 0
    sum = 0
    for j in best[i]:
        selectedItem = short.loc[short['user_id'] == j]
        y += mas[i][j]*selectedItem['category_1_bin'].values[0]
        print(mas[i][j], selectedItem['category_1_bin'].values[0])
        sum += mas[i][j]
    y = y/sum
    YPredCol.append(y)
```

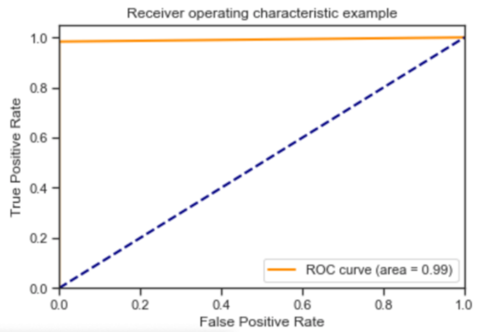
```
0.9999999927953996 0
0.9999999967979537 0
0.9999999967979537 0
0.9999999988647608 0
1.0000000000000002 0
0.9999993978776253 0
0.9999999969726886 0
```

]:
```
YColFIlt = []
for i in YPredCol:
    if i >=0.9:
        YColFIlt.append(1)
    else:
        YColFIlt.append(0)
```

]:
```
accFil = accuracy_score(YTrueFil, YColFIlt)
```

]:
```
preFil = precision_score(YTrueFil, YColFIlt)
recFil = recall_score(YTrueFil, YColFIlt)
```

]:
```
draw_roc_curve(YTrueFil, YColFIlt, pos_label=1, average='micro')
```



pand output; double click to hide output

]:
```
df_results.append({'Method':"Filtering", 'Accuracy':accFil,'Precision':preFil, 'Recall':recFil} , ignore_index=True)
```

]:

	Method	Accuracy	Precision	Recall
0	7 clusters	0.6008	0.428373	0.344134
1	10 clusters	0.6020	0.433511	0.364246
2	15 clusters	0.6284	0.466135	0.261453
3	Filtering	0.9940	1.000000	0.983240