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:

Костян Алина Алексеевна
Группа ИУ5-21М

____

Москва 2021

Задание:

Выбрать произвольный набор данных (датасет), предназначенный для построения рекомендательных моделей.

Опираясь на материалы лекции, сформировать рекомендации для одного пользователя (объекта) двумя произвольными способами.

Сравнить полученные рекомендации (если это возможно, то с применением метрик).

Текст программы:

```
In [7]: data.head()
```

Out[7]:

	User-ID	ISBN	Book-Rating	Location	Age	Book-Title	Book-Author	Year-Of-Publication	Publisher	Image-URL-S	
0	276725	034545104X	0	tyler, texas, usa	NaN	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	http://images.amazon.com/images/P/034545104X.0...	http://images.amazon.com
1	2313	034545104X	5	cincinnati, ohio, usa	23.0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	http://images.amazon.com/images/P/034545104X.0...	http://images.amazon.com
2	6543	034545104X	0	strafford, missouri, usa	34.0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	http://images.amazon.com/images/P/034545104X.0...	http://images.amazon.com
3	8680	034545104X	5	st. charles county, missouri, usa	2.0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	http://images.amazon.com/images/P/034545104X.0...	http://images.amazon.com
4	10314	034545104X	9	beaverton, oregon, usa	NaN	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	http://images.amazon.com/images/P/034545104X.0...	http://images.amazon.com

```
In [8]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1031136 entries, 0 to 1031135
Data columns (total 12 columns):
 #   Column               Non-Null Count    Dtype
---  ------               --------------    -----
 0   User-ID              1031136 non-null  int64
 1   ISBN                 1031136 non-null  object
 2   Book-Rating          1031136 non-null  int64
 3   Location             1031136 non-null  object
 4   Age                  753301 non-null   float64
 5   Book-Title           1031136 non-null  object
 6   Book-Author          1031135 non-null  object
 7   Year-Of-Publication  1031136 non-null  object
 8   Publisher            1031134 non-null  object
 9   Image-URL-S          1031136 non-null  object
 10  Image-URL-M          1031136 non-null  object
 11  Image-URL-L          1031132 non-null  object
dtypes: float64(1), int64(2), object(9)
memory usage: 102.3+ MB
```

In [9]:
```
print('Number of books: ', data['ISBN'].nunique())
print('Number of users: ',data['User-ID'].nunique())
```

```
Number of books:  270151
Number of users:  92106
```

In [10]:
```
median = data["Age"].median()
std = data["Age"].std()
is_null = data["Age"].isnull().sum()
rand_age = np.random.randint(median - std, median + std, size = is_null)
age_slice = data["Age"].copy()
age_slice[pd.isnull(age_slice)] = rand_age
data["Age"] = age_slice
data["Age"] = data["Age"].astype(int)
```

In [11]:
```
data['Book-Rating'] = data['Book-Rating'].replace(0, None)
```

In [12]:
```
data[['Book-Author', 'Publisher']] = data[['Book-Author', 'Publisher']].fillna('Unknown')
```

In [13]:
```
data = data.dropna(axis=0, how='any')
data = data.drop(['Image-URL-S', 'Image-URL-M', 'Image-URL-L'], axis=1)
```

In [14]:
```
data.isnull().sum()
```

Out[14]:
```
User-ID                0
ISBN                   0
Book-Rating            0
Location               0
Age                    0
Book-Title             0
Book-Author            0
Year-Of-Publication    0
Publisher              0
dtype: int64
```

In [15]:
```
data['Country'] = data['Location'].apply(lambda row: str(row).split(',')[-1])
data = data.drop('Location', axis=1)
data['Country'].head()
```

Out[15]:
```
0     usa
1     usa
2     usa
3     usa
4     usa
Name: Country, dtype: object
```

In [16]:
```
data['Year-Of-Publication'] = pd.to_numeric(data['Year-Of-Publication'])
```

In [17]:
```
df = data
```

In [18]:
```
df = df[df['Book-Rating'] >= 6]
df.groupby('ISBN')['User-ID'].count().describe()
```

Out[18]:
```
count    228988.000000
mean          3.728409
std          12.416574
min           1.000000
25%           1.000000
50%           1.000000
75%           3.000000
max        1206.000000
Name: User-ID, dtype: float64
```

In [19]:
```
title = df['Book-Title'].values
author = df['Book-Author'].values
publisher = df['Publisher'].values
ISBN = df['ISBN'].values
rating = df['Book-Rating'].values
```

In [20]:
```
%%time
tfidfv = TfidfVectorizer()
title_matrix = tfidfv.fit_transform(title)
author_matrix = tfidfv.fit_transform(author)
publisher_matrix = tfidfv.fit_transform(publisher)
ISBN_matrix = tfidfv.fit_transform(ISBN)
```

```
CPU times: user 15 s, sys: 284 ms, total: 15.2 s
Wall time: 15.3 s
```

Фильтрация на основе содержания

In [21]:
```
class SimpleKNNRecommender:

    def __init__(self, X_matrix, X_ids, X_title, X_overview):
        """
        Входные параметры:
        X_matrix - обучающая выборка (матрица объект-признак)
        X_ids - массив идентификаторов объектов
        X_title - массив названий объектов
        X_overview - массив описаний объектов
        """
        #Сохраняем параметры в переменных объекта
        self._X_matrix = X_matrix
        self.df = pd.DataFrame(
            {'id': pd.Series(X_ids, dtype='int'),
            'title': pd.Series(X_title, dtype='str'),
            'overview': pd.Series(X_overview, dtype='str'),
            'dist': pd.Series([], dtype='float')})


    def recommend_for_single_object(self, K: int, \
                X_matrix_object, cos_flag = True, manh_flag = False):
        """
        Метод формирования рекомендаций для одного объекта.
        Входные параметры:
        K - количество рекомендуемых соседей
        X_matrix_object - строка матрицы объект-признак, соответствующая объекту
        cos_flag - флаг вычисления косинусного расстояния
        manh_flag - флаг вычисления манхэттэнского расстояния
        Возвращаемое значение: K найденных соседей
        """

        scale = 1000000
        # Вычисляем косинусную близость
        if cos_flag:
            dist = cosine_similarity(self._X_matrix, X_matrix_object)
            self.df['dist'] = dist * scale
            res = self.df.sort_values(by='dist', ascending=False)
            # Не учитываем рекомендации с единичным расстоянием,
            # так как это искомый объект
            res = res[res['dist'] < scale]

        else:
            if manh_flag:
                dist = manhattan_distances(self._X_matrix, X_matrix_object)
            else:
                dist = euclidean_distances(self._X_matrix, X_matrix_object)
            self.df['dist'] = dist * scale
            res = self.df.sort_values(by='dist', ascending=True)
            # Не учитываем рекомендации с единичным расстоянием,
            # так как это искомый объект
            res = res[res['dist'] > 0.0]

        # Оставляем K первых рекомендаций
        res = res.head(K)
        return res
```

In [22]:
```
author_ind = 54
author[author_ind]
```

Out[22]: 'Nicholas Sparks'

In [23]:
```
sparks_matrix = author_matrix[author_ind]
```

In [24]:
```
skr1 = SimpleKNNRecommender(author_matrix, rating, title, author)
rec1 = skr1.recommend_for_single_object(15, sparks_matrix)
rec1
```

Out[24]:

	id	title	overview	dist
442981	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	679127.939310
442976	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	679127.939310
442980	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	679127.939310
442979	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	679127.939310
442978	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	679127.939310
442977	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	679127.939310
328753	6	The Discovery of Animal Behaviour	John Sparks	617606.995898
623204	9	The Last of the Cockleshell Heroes: A World Wa...	William Sparks	570049.319484
822969	7	The Next Archaeology Workbook	Nicholas David	526087.742446
516949	9	Cook: The Extraordinary Voyages of Captain Jam...	Nicholas Thomas	496039.823116
589733	10	The Elephant Man	Christine Sparks	485967.020337
852180	7	Elephant Man	Christine Sparks	485967.020337
589732	10	The Elephant Man	Christine Sparks	485967.020337
281592	7	Veronica	Nicholas Christopher	475879.754871
390227	9	Veronica: A Novel	Nicholas Christopher	475879.754871

In [25]:
```
rec2 = skr1.recommend_for_single_object(15, sparks_matrix, cos_flag = False)
rec2
```

Out[25]:

	id	title	overview	dist
442979	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	801089.334207
442976	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	801089.334207
442978	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	801089.334207
442977	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	801089.334207
442980	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	801089.334207
442981	10	A Lifelong Passion: Nicholas and Alexandra	Nicholas	801089.334207
328753	6	The Discovery of Animal Behaviour	John Sparks	874520.444703
623204	9	The Last of the Cockleshell Heroes: A World Wa...	William Sparks	927308.665457
822969	7	The Next Archaeology Workbook	Nicholas David	973562.794640
456109	8	The Sensuous Woman	J	1000000.000000
814693	6	The Power of Five (W.I.T.C.H., 1)	W.i.t.c.h.	1000000.000000
814692	8	The Power of Five (W.I.T.C.H., 1)	W.i.t.c.h.	1000000.000000
578909	7	Quilting with the Muppets: 15 Fun and Creative...	N	1000000.000000
814691	8	W.I.T.C.H. Chapter Book: The Four Dragons - Bo...	W.i.t.c.h.	1000000.000000
765435	10	Getting Even: Making O	X	1000000.000000

Метод на основе сингулярного разложения

In [26]:
```
def create_utility_matrix(data):
    itemField = 'Book-Title'
    userField = 'User-ID'
    valueField = 'Book-Rating'

    userList = data[userField].tolist()
    itemList = data[itemField].tolist()
    valueList = data[valueField].tolist()

    users = list(set(userList))
    items = list(set(itemList))

    users_index = {users[i]: i for i in range(len(users))}
    pd_dict = {item: [0.0 for i in range(len(users))] for item in items}

    for i in range(0,data.shape[0]):
        item = itemList[i]
        user = userList[i]
        value = valueList[i]
        pd_dict[item][users_index[user]] = value

    X = pd.DataFrame(pd_dict)
    X.index = users

    itemcols = list(X.columns)
    items_index = {itemcols[i]: i for i in range(len(itemcols))}

    return X, users_index, items_index
```

In [27]:
```
mini_df = df[0:500]
```

In [28]:
```
%%time
user_item_matrix, users_index, items_index = create_utility_matrix(mini_df)
```

```
CPU times: user 2.21 ms, sys: 133 µs, total: 2.35 ms
Wall time: 2.32 ms
```

In [29]:
```
user_item_matrix
```

Out[29]:

	Lightning	A Painted House	The Amsterdam Connection : Level 4 (Cambridge English Readers)	Manhattan Hunt Club	Flesh Tones: A Novel	The Notebook	Les Particules Elementaires
63507	0.0	0.0	0.0	0.0	0.0	8.0	0.0
92184	0.0	8.0	0.0	0.0	0.0	0.0	0.0
235560	9.0	0.0	0.0	0.0	0.0	0.0	0.0
8234	0.0	7.0	0.0	0.0	0.0	0.0	0.0
145451	0.0	0.0	0.0	0.0	7.0	0.0	0.0
...	...	...	...	...	...	...	...
161765	0.0	0.0	0.0	9.0	0.0	0.0	0.0
135149	9.0	0.0	0.0	0.0	0.0	0.0	0.0
20462	8.0	0.0	0.0	0.0	0.0	0.0	0.0
30711	0.0	0.0	0.0	0.0	0.0	6.0	0.0
172030	8.0	0.0	0.0	0.0	0.0	0.0	0.0

470 rows × 7 columns

In [30]:
```
user_item_matrix__test = user_item_matrix.iloc[469]
user_item_matrix__train = user_item_matrix.iloc[:470]
```

In [31]:
```
%%time
U, S, VT = np.linalg.svd(user_item_matrix__train.T)
V = VT.T
```

```
CPU times: user 4.41 ms, sys: 2.41 ms, total: 6.82 ms
Wall time: 2.5 ms
```

In [32]:
```
Sigma = np.diag(S)
```

In [33]:
```
r=3
Ur = U[:, :r]
Sr = Sigma[:r, :r]
Vr = V[:, :r]
```

In [34]:
```
test_user = np.mat(user_item_matrix__test.values)
test_user.shape, test_user
```

Out[34]:
```
((1, 7), matrix([[8., 0., 0., 0., 0., 0., 0.]]))
```

In [35]:
```
tmp = test_user * Ur * np.linalg.inv(Sr)
tmp
```

Out[35]:
```
matrix([[-0.013692  ,  0.07842731, -0.02007541]])
```

In [36]:
```
test_user_result = np.array([tmp[0,0], tmp[0,1], tmp[0,2]])
test_user_result
```

Out[36]:
```
array([-0.013692  ,  0.07842731, -0.02007541])
```

In [37]:
```
# Вычисляем косинусную близость между текущим пользователем
# и остальными пользователями
cos_sim = cosine_similarity(Vr, test_user_result.reshape(1, -1))
cos_sim[:10]
```

Out[37]:
```
array([[-0.05145611],
       [-0.02704594],
       [ 1.        ],
       [-0.02704594],
       [ 0.87936934],
       [-0.02704594],
       [ 1.        ],
       [-0.02704594],
       [ 1.        ],
       [ 0.87936934]])
```

In [38]:
```
# Преобразуем размерность массива
cos_sim_list = cos_sim.reshape(-1, cos_sim.shape[0])[0]
cos_sim_list[:10]
```

Out[38]:
```
array([-0.05145611, -0.02704594,  1.        , -0.02704594,  0.87936934,
       -0.02704594,  1.        , -0.02704594,  1.        ,  0.87936934])
```

In [39]:
```
# Находим наиболее близкого пользователя
recommended_user_id = np.argsort(-cos_sim_list)[0]
recommended_user_id
```

Out[39]:
```
392
```

In [40]:
```
# Получение названия фильма
userId_list = list(user_item_matrix.columns)
def book_name_by_user(ind):
    try:
        userId = userId_list[ind]
        flt_links = mini_df[mini_df['User-ID'] == userId]
        #tmdbId = int(flt_links['tmdbId'].values[0])
        #md_links = df_md[df_md['id'] == tmdbId]
        res = mini_df['Book-Title'].values[0]
        return res
    except:
        return ''
```

In [41]:
```
i=1
for idx, item in enumerate(np.ndarray.flatten(np.array(test_user))):
    if item > 0:
        book_title = book_name_by_user(idx)
        print('{} - {} - {}'.format(idx, book_title, item))
```

```
0 - Flesh Tones: A Novel - 8.0
```