Сопроводительная документация по задаче

Анализ полученного датасета

1. Выявление ошибок. Список некорректных url просмотров из dataset_news_1.xlsx

```
mos.ru/news/item/89421073/ /
mos.ru/news/item/9468/
mos.ru/news/item/94670073/ /
mos.ru/news/item/94501073/душ/
mos.ru/news/item/89957073/ Их/
mos.ru/news/item/94852073/%5c/
mos.ru/news/item/94479073/ (https:/app.aif.ru/owa/redir.aspx/
mos.ru/news/item/94792073/ /
mos.ru/news/item/94897073/+/
mos.ru/news/item/94953073/ /
mos.ru/news/item/91919073/-/
```

2. Анализ полученных данных:

- 239 пользователей на 5812 новости для 26 446 просмотров

Протестированные гипотезы и алгоритм работы решения для рекомендательной системы

In [71]:

```
import pandas as pd
import numpy as np
import json
import datetime
from scipy.sparse import csr_matrix
from implicit.nearest_neighbours import ItemItemRecommender
from implicit.als import AlternatingLeastSquares
from implicit.evaluation import precision_at_k, mean_average_precision_at_
k
```

Загружаем данные

In [18]:

```
def get_news_id_from_url(url: str) -> int:
    """
    id из url
    """
    parts = url.split('/')
    try:
        return int(parts[-2])
    except Exception as err:
        for part in parts:
            if '073' in part:  # Опытным путем выявлено, что битые урлы
```

```
                # только для типа 073, поэтому просто решила вытащить таки
е
                return int(part)
        return 0


df_views = pd.read_excel('/app/data/dataset_news_1.xlsx')
df_news = pd.read_json('/app/data/news.json', encoding="utf_8_sig")
df_views['news_id'] = df_views['url_clean'].apply(get_news_id_from_url)
merged = df_views.merge(df_news, left_on='news_id', right_on='id')
```

In [97]:

```
final_df = merged.drop(['importance', 'is_deferred_publication', 'status',
'ya_rss', 'active_from',
                        'active_to', 'search', 'display_image', 'icon_id',
'canonical_url', 'canonical_updated_at',
                        'is_powered', 'has_image', 'attach', 'active_from_t
imestamp', 'active_to_timestamp',
                        'image', 'counter', 'preview_text', 'images'],
                       axis=1)
final_df['title_age'] = (pd.Timestamp.now() - final_df['published_at']).dt
.days
final_df['age_param'] = 1 / final_df['title_age']
users, items, interactions = final_df.user_id.nunique(), final_df.id.nuniq
ue(), final_df.shape[0]
print('# users: ', users)
print('# items: ', items)
print('# interactions: ', interactions)
final_df = final_df[['user_id', 'news_id', 'date_time', 'age_param']]
final_df.head()
```

```
# users:  239
# items:  5809
# interactions:  26442
```

Out[97]:

	user_id	news_id	date_time	age_param
0	1	94006073	2021-08-01 18:51:19	0.011905
1	2	94006073	2021-08-04 13:08:19	0.011905
2	3	94006073	2021-08-29 12:40:07	0.011905
3	6	94006073	2021-08-02 09:04:55	0.011905
4	11	94006073	2021-08-02 17:16:23	0.011905

age_param - величина, обратно пропорциональная количеству дней с даты публикации новости. Это значение мы используем для определения актуальности новости на момент просмотра.

Разделяем данные на тренировочные и тестовые. В train берем 3 недели августа от даты просмотра, остальное в test.

In [98]:

```
test_size_days = 20

data_train = final_df[final_df['date_time'].dt.day < final_df['date_time']
.dt.day.min() + test_size_days]
data_test = final_df[final_df['date_time'].dt.day >= final_df['date_time']
.dt.day.min() + test_size_days]
print("Количество просмотров в train: ", data_train.shape[0])
print("Количество просмотров в test: ", data_test.shape[0])
```

```
Количество просмотров в train:  20483
Количество просмотров в test:  5959
```

Готовим результирующий сет данных для проверки рекомендаций.

In [99]:

```
result = data_test.groupby('user_id')['news_id'].unique().reset_index()
result.columns = ['user_id', 'history']
result['history'] = result['history'].apply(lambda x: list(x))
result.head(5)
```

Out[99]:

	user_id	history
0	2	[94339073, 94351073]
1	3	[94006073, 94108073, 94642073, 94860073, 75790...
2	4	[94953073, 95030073, 95023073, 95149073, 95151...
3	5	[94482073, 94953073, 95149073, 94898073, 75970...
4	6	[94953073, 95030073, 95149073, 95076073, 95148...

Подготовливаем матрицы для обучения и тестирования модели. user_item_matrix_test - матрица, которая содержит все исходные данные для проверки модели. user_item_matrix - тренировачная матрица, которая содержит только данные для обучения. Размеры матрицы соответствуют количеству уникальных пользователей к количеству уникальных новостей. На пересечении в качестве значимого параметра используем age_param (актуальность новости в момент получения рекомендаций). Если у новости не было просмотров, то присваиваем значение параметра 0.

In [100]:

```
user_item_matrix_test = pd.pivot_table(final_df,
                                       index='user_id', columns='news_id',
                                       values='age_param',
                                       fill_value=0
                                      )
user_item_matrix = user_item_matrix_test.copy(deep=True)
for index, row in data_test.iterrows():
    user_id = row['user_id']
    news_id = row['news_id']
    user_item_matrix.loc[user_id, news_id] = 0


user_item_matrix = user_item_matrix.astype(float)
```

```
sparse_user_item = csr_matrix(user_item_matrix).T.tocsr()
sparse_user_item_test = csr_matrix(user_item_matrix_test).T.tocsr()

print("Размер train матрицы: ", user_item_matrix.shape)
print("Размер test матрицы: ", user_item_matrix_test.shape)

user_item_matrix.describe()
```

```
Размер train матрицы:  (239, 5809)
Размер test матрицы:  (239, 5809)
```

Out[100]:

news_id	179050	1221050	1261050	1319050	1918050	1931050	19880
count	239.000000	239.0	239.000000	239.000000	239.0	239.000000	239
mean	0.000001	0.0	0.000001	0.000001	0.0	0.000002	(
std	0.000017	0.0	0.000021	0.000022	0.0	0.000025	(
min	0.000000	0.0	0.000000	0.000000	0.0	0.000000	(
25%	0.000000	0.0	0.000000	0.000000	0.0	0.000000	(
50%	0.000000	0.0	0.000000	0.000000	0.0	0.000000	(
75%	0.000000	0.0	0.000000	0.000000	0.0	0.000000	(
max	0.000270	0.0	0.000332	0.000334	0.0	0.000381	(

8 rows × 5809 columns

In [102]:

```
user_item_matrix_test.describe()
```

Out[102]:

news_id	179050	1221050	1261050	1319050	1918050	1931050
count	239.000000	239.000000	239.000000	239.000000	239.000000	239.000000
mean	0.000001	0.000001	0.000001	0.000001	0.000002	0.000002
std	0.000017	0.000021	0.000021	0.000022	0.000025	0.000025
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	0.000270	0.000330	0.000332	0.000334	0.000382	0.000381

8 rows × 5809 columns

Функция для получения рекомендаций от обученной модели.

In [103]:

```
user_ids = list(user_item_matrix_test.index.values)
```

```
news_ids = list(user_item_matrix_test.columns.values)
def recomend_test_user(user_id, model):
    user_index = user_ids.index(user_id)
    recommendations = model.recommend(user_index, sparse_user_item_test, N
=20)
    result = [news_ids[x[0]] for x in recommendations]
    return result
```

Обучение модели ItemItemRecommender

Алгоритм на основе метода ближайших соседей.

[модель ItemItemRecommender]

In [104]:

```
%%time

model_iir = ItemItemRecommender(K=20, num_threads=4)
model_iir.fit(csr_matrix(user_item_matrix).T, show_progress=True)

model_iir.recommend(12, sparse_user_item, N=20)
```

```
CPU times: user 262 ms, sys: 256 ms, total: 518 ms
Wall time: 245 ms
```

Out[104]:

```
[(4999, 8.567021181711429e-09),
 (5003, 8.567021181711429e-09),
 (4983, 8.457187576817692e-09),
 (4993, 8.457187576817692e-09),
 (4997, 8.457187576817692e-09),
 (4954, 8.350134569516203e-09),
 (4942, 8.350134569516203e-09),
 (4903, 8.24575788739725e-09),
 (4913, 8.24575788739725e-09),
 (663, 8.143958407305925e-09),
 (4813, 8.143958407305925e-09),
 (4841, 8.143958407305925e-09),
 (4767, 8.044641841363172e-09),
 (4773, 8.044641841363172e-09),
 (4810, 8.044641841363172e-09),
 (4590, 7.853102749902143e-09),
 (4708, 7.853102749902143e-09),
 (4728, 7.853102749902143e-09),
 (4734, 7.853102749902143e-09),
 (4729, 7.760713305785647e-09)]
```

Обучение модели ALS (Alternating Least Squares)

Алгоритм наименьших квадратов

[модель AlternatingLeastSquares]

In [115]:

```
%%time
model_als = AlternatingLeastSquares(factors=100, #k f
                                    regularization=0.001,
                                    iterations=15,
                                    calculate_training_loss=True,
                                    num_threads=4)

model_als.fit(sparse_user_item, show_progress=True)

model_als.recommend(12, user_items=sparse_user_item, N=20)
```

```
CPU times: user 26.9 s, sys: 32.5 s, total: 59.4 s
Wall time: 5.37 s
```

Out[115]:

```
[(4723, 0.13660578),
 (5049, 0.12596613),
 (698, 0.105814315),
 (685, 0.089494444),
 (5305, 0.08479826),
 (696, 0.07202895),
 (4934, 0.070641376),
 (678, 0.07032433),
 (695, 0.068552166),
 (5304, 0.06642769),
 (5250, 0.059926048),
 (684, 0.05811741),
 (4879, 0.058020774),
 (5192, 0.057265192),
 (5173, 0.05361002),
 (4817, 0.048172574),
 (4909, 0.046892427),
 (4867, 0.04301165),
 (5054, 0.042675257),
 (4771, 0.038593516)]
```

In [106]:

```
result['itemitem'] = result.apply(lambda x: recomend_test_user(x['user_id'
], model_iir), axis='columns')
result['diff_iir'] = result.apply(lambda x: len(set(x["history"]) & set(x[
"itemitem"])), axis='columns')
result['als'] = result.apply(lambda x: recomend_test_user(x['user_id'], mo
del_als), axis='columns')
result['diff_als'] = result.apply(lambda x: len(set(x["history"]) & set(x[
"als"])), axis='columns')

result.head()
```

Out[106]:

	user_id	history	itemitem	diff_iir	als	diff_als
0	2	[94339073, 94351073]	[94643073, 94645073, 94614073, 94633073, 75670...	0	[94292073, 94860073, 94006073, 94115073, 94266...	0
		[94006073,	[94860073,		[94801073,	

1	3	94108073, 94642073, 94860073, 75790...	94765073, 94865073, 94724073, 94852...	2	94482073, 94469073, 94779073, 94754...	0
2	4	[94953073, 95030073, 95023073, 95149073, 95151...	[94702073, 94659073, 94705073, 94681073, 94688...	0	[94702073, 94707073, 94701073, 94779073, 94638...	0
3	5	[94482073, 94953073, 95149073, 94898073, 75970...	[94900073, 94874073, 94913073, 94875073, 94876...	0	[94792073, 7552050, 94415073, 94339073, 946390...	0
4	6	[94953073, 95030073, 95149073, 95076073, 95148...	[94702073, 94659073, 94705073, 94681073, 94688...	0	[94363073, 94634073, 94638073, 94696073, 94415...	0

In [107]:

```
result.describe()
```

Out[107]:

	user_id	diff_iir	diff_als
count	163.000000	163.000000	163.000000
mean	131.349693	0.226994	0.067485
std	77.642146	0.580555	0.370666
min	2.000000	0.000000	0.000000
25%	68.500000	0.000000	0.000000
50%	129.000000	0.000000	0.000000
75%	190.000000	0.000000	0.000000
max	275.000000	3.000000	3.000000

In [114]:

```
positive_result_iir_count = result[result['diff_iir'] > 0].shape[0]
positive_result_als_count = result[result['diff_als'] > 0].shape[0]
print("Количество попаданий для модели IIR: ", positive_result_iir_count)
print("Количество попаданий для модели ALS: ", positive_result_als_count)
```

```
Количество попаданий для модели IIR:  26
Количество попаданий для модели ALS:  7
```

In [109]:

```
map_iir = mean_average_precision_at_k(model_iir, sparse_user_item.T, spars
e_user_item_test.T, K=5)
map_als = d = mean_average_precision_at_k(model_als, sparse_user_item.T, s
parse_user_item_test.T, K=5)
print("mean average precision at k for model ItemItemRecommender: ", map_i
ir)
print("mean average precision at k for model ALS: ", map_als)
```

```
mean average precision at k for model ItemItemRecommender:
0.01444909344490935
mean average precision at k for model ALS:  0.008647140864714
088
```

In [110]:

```
precision_iir = precision_at_k(model_iir, sparse_user_item.T, sparse_user_
item_test.T, K=5)
precision_als = precision_at_k(model_als, sparse_user_item.T, sparse_user_
item_test.T, K=5)
print("precision at k for model ItemItemRecommender: ", precision_iir)
print("precision at k for model ALS: ", precision_als)
```

```
precision at k for model ItemItemRecommender:  0.025941422594
14226
precision at k for model ALS:  0.015899581589958158
```

Модель ItemItemRecommender по текущим показателям выигрывает у ALS, но при больших данных и более глубоком погружении в тематику можно достичь лучших результатов. Помимо этого можно также использовать гибридный тип (на базе анализа контента и метода ближайших соседей) коллаборативной фильтрации, который позволит улучшить качество рекомендательной системы.

Описание алгоритма для авторазметки новостей

Алгоритм разметки сделан на основе меры TF-IDF. Он выбирает наиболее весомые слова на основе частоты употребления в документе в сравнении с полным корпусом. Полный корпус составляется на основе всех новостей, их тегов и сфер, исключая стоп-слова. Результатом алгоритма является набор тегов и сфер для переданного текста новости.

Функции для обработки текста:

- get_text_on_pattern_replacement_func - очистка от html-тегов
- get_lst_of_normalized_tokens_without_stopwords - нормализация слов и очистка от стоп-слов

Функции для создания корпуса всех доступных материалов, тегов и сфер

Основные функции алгоритма:

- get_result_tag_and_spheres_for_title_preview_fulltext - функция для получения результатов
- compute_idf - функция для расчета IDF
- compute_tf - функция для расчета TF
- get_named_objects_without_stopwords - функция для получения именованных объектов для обогощения результатов (выдает адреса, названия, имена, организации)

Используемые технологии:

- nltk
- pymorphy2
- natasha