

Министерство науки и высшего образования
Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования «Рыбинский государственный
авиационный технический университет имени П. А. Соловьева»

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И СИСТЕМЫ УПРАВЛЕНИЯ

Математического и программного обеспечения электронных
вычислительных средств

ОТЧЁТ

по дисциплине:

«Методы и алгоритмы анализа данных»

на тему:

«Рекомендательные системы»

Выполнил: студент группы ИВМ-24

Морозов А. А.

Руководитель: ассистент

Вязниковцев Д. А.

Рыбинск 2024

Содержание

Цель работы	3
1 ALS факторизация.....	4
2 Поиск похожих фильмов	9
3 Поиск похожего пользователя	10
4 Оценка качества метрикой $NDCG$	12
Вывод.....	14

Цель работы

Целями данной лабораторной работы являются:

- изучение алгоритма ALS для поиска похожих фильмов;
- реализация подсчета метрики NDCG;
- исследование влияния размерности скрытых представлений на работу алгоритма ALS.

1 ALS факторизация

В начале необходимо загрузить датасеты с фильмами, пользователями и рейтингами из архива *ml-1m.zip*:

```
with zipfile.ZipFile("ml-1m.zip", "r") as z:
    # parse movies
    with z.open("ml-1m/movies.dat") as m:
        for line in m:
            MovieID, Title, Genres = line.decode('iso-8859-1').strip().split("::")
            MovieID = int(MovieID)
            Genres = Genres.split("|")
            if Title == "Star Wars: Episode V - The Empire Strikes Back (1980)":
                StarWarsID = MovieID
                print(f"MovieID у фильма Звёздные войны 5: {StarWarsID}")
                movies[MovieID] = {"Title": Title, "Genres": Genres}

    # parse users
    with z.open("ml-1m/users.dat") as m:
        fields = ["UserID", "Gender", "Age", "Occupation", "Zip-code"]
        for line in m:
            row = list(zip(fields, line.decode('iso-8859-1').strip().split("::")))
            data = dict(row[1:])
            data["Occupation"] = int(data["Occupation"])
            users[int(row[0][1])] = data

    # parse ratings
    with z.open("ml-1m/ratings.dat") as m:
        for line in m:
            UserID, MovieID, Rating, Timestamp = line.decode('iso-8859-1').strip().split("::")
            UserID = int(UserID)
            MovieID = int(MovieID)
            Rating = int(Rating)
            Timestamp = int(Timestamp)
```

```
ratings[UserID].append((MovieID, Rating,
datetime.datetime.fromtimestamp(Timestamp)))
```

После получения данных из zip-файла необходимо разделить данные на тестовые и обучающие по дате оценки фильма (80% у обучающей выборки, а 20% у тестовой):

```
# train-test split
times = []
for user_ratings in ratings.values():
    times.extend([x[2] for x in user_ratings])
times = sorted(times)
threshold_time = times[int(0.8 * len(times))]

train = []
test = []
for user_id, user_ratings in ratings.items():
    train.extend((user_id, rating[0], rating[1] / 5.0) for rating in
user_ratings if rating[2] <= threshold_time)
    test.extend((user_id, rating[0], rating[1] / 5.0) for rating in user_ratings
if rating[2] > threshold_time)
print("ratings in train:", len(train))
print("ratings in test:", len(test))
train_by_user = defaultdict(list)
test_by_user = defaultdict(list)
for u, i, r in train:
    train_by_user[u].append((i, r))

for u, i, r in test:
    test_by_user[u].append((i, r))
```

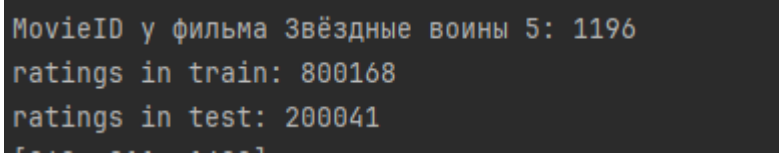
```

train_by_item = defaultdict(list)
for u, i, r in train:
    train_by_item[i].append((u, r))

n_users = max([e[0] for e in train]) + 1
n_items = max([e[1] for e in train]) + 1

```

На рисунке 1.1 показано отображение консоли после выполнения кода выше.



```

MovieID у фильма Звёздные войны 5: 1196
ratings in train: 800168
ratings in test: 200041

```

Рисунок 1.1 – Вывод консоли после разбивки данных

Теперь можно приступить к реализации *ALS* метода:

```

# Реализация ALS
np.random.seed(0)
LATENT_SIZE = 10
N_ITER = 20

# регуляризаторы
lambda_p = 0.2
lambda_q = 0.001

# латентные представления
p = 0.1 * np.random.random((n_users, LATENT_SIZE))

```

```
q = 0.1 * np.random.random((n_items, LATENT_SIZE))
```

```
def compute_p(p, q, train_by_user):
```

```
    for u, rated in train_by_user.items():
```

```
        rated_items = [i for i, _ in rated]
```

```
        rated_scores = np.array([r for _, r in rated])
```

```
        Q = q[rated_items, :]
```

```
        A = (Q.T).dot(Q)
```

```
        d = (Q.T).dot(rated_scores)
```

```
        p[u, :] = np.linalg.solve(lambda_p * len(rated_items) *
np.eye(LATENT_SIZE) + A, d)
    return p
```

```
def compute_q(p, q, train_by_item):
```

```
    for i, rated in train_by_item.items():
```

```
        rated_users = [j for j, _ in rated]
```

```
        rated_scores = np.array([s for _, s in rated])
```

```
        P = p[rated_users, :]
```

```
        A = (P.T).dot(P)
```

```
        d = (P.T).dot(rated_scores)
```

```
        q[i, :] = np.linalg.solve(lambda_q * len(rated_users) *
np.eye(LATENT_SIZE) + A, d)
    return q
```

```
def train_error_mse(predictions):
```

```
    return np.mean([(predictions[u, i] - r) ** 2 for u, i, r in train])
```

```
def test_error_mse(predictions):
```

```
    return np.mean([(predictions[u, i] - r) ** 2 for u, i, r in test])
```

```
for iter in range(N_ITER):  
    p = compute_p(p, q, train_by_user)  
    q = compute_q(p, q, train_by_item)  
  
    predictions = p.dot(q.T)  
  
    print(iter, train_error_mse(predictions), test_error_mse(predictions))
```

С помощью этого кода происходит обновление латентных представлений пользователей и их оценок, а также поиск оптимальных значений с помощью *MSE*. На 19 итерации получится такой результат: 0.024542448316282744 0.08630578168733996.

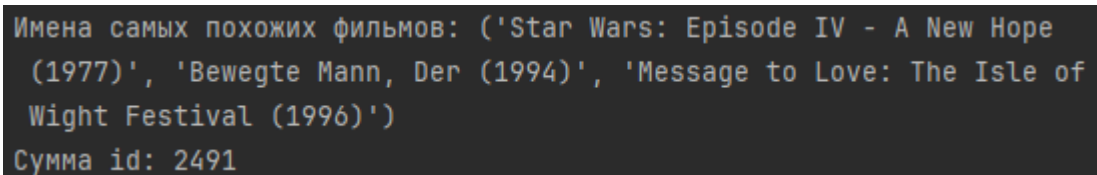
2 Поиск похожих фильмов

При разделении данных на 2 выборки был найден *MovieID* у фильма с названием «*Star Wars: Episode V - The Empire Strikes Back* (1980)» (рисунок 1.1.). С помощью матрицы в переменной *q* необходимо найти 3 похожих фильма:

```
skal = {}
for i in range(len(q)):
    if i != StarWarsID:
        skal[i] = np.dot(q[i], q[StarWarsID])

top_3 = [key for key, value in sorted(skal.items(), key=lambda x: x[1],
reverse=True)[:3]]
print(top_3)
print(f'Имена самых похожих фильмов: {movies[top_3[0]]['Title'],
movies[top_3[1]]['Title'], movies[top_3[2]]['Title']}')
print(f'Сумма id: {sum(top_3)}')
```

На рисунке 2.1 показано отображение консоли после выполнения кода выше.



```
Имена самых похожих фильмов: ('Star Wars: Episode IV - A New Hope
(1977)', 'Bewegte Mann, Der (1994)', 'Message to Love: The Isle of
Wight Festival (1996)')
Сумма id: 2491
```

Рисунок 2.1 – Вывод консоли после поиска похожих фильмов

3 Поиск похожего пользователя

Необходимо найти для пользователя с *ID* 5472 самого похожего другого пользователя и определить какое количество одних и тех же фильмов они просмотрели:

```
skalP = {}
for i in range(len(p)):
    if i != 5472:
        skalP[i] = np.dot(p[i], p[5472])

top = [key for key, value in sorted(skalP.items(), key=lambda x: x[1],
reverse=True)]
# print(top)
print(f"Имя самого похожего человека по просмотренным фильмам с 5472:
{users[top[0]]}")
films1 = ratings[5472]
films2 = ratings[top[0]]
cnt = 0
for i in films1:
    for j in films2:
        if i[0] == j[0]:
            cnt += 1
print(cnt)
```

На рисунке 3.1 показано отображение консоли после выполнения кода выше.

```
Имя самого похожего человека по просмотренным фильмам с 5472: {'Gender':  
  'M', 'Age': '45', 'Occupation': 17, 'Zip-code': '97330'}  
27  
500-1-50-550754000101
```

Рисунок 3.1 – Вывод консоли после поиска похожего пользователя

В итоге 2 пользователя просмотрели 27 одних и тех же фильмов.

4 Оценка качества метрикой *NDCG*

NDCG – это метрика, которая используется для оценки качества ранжирования результатов. Нам уже даны 19 элементов с их релевантностью и количество элементов, которое будет рассматриваться (таких будет 5). Для использования этой метрики воспользуемся кодом:

```
def DCG_k(ratings_list, k):
    """
    ratings_list: np.array(n_items,)
    k: int
    """
    summa = 0
    for i in range(k):
        summa += ((2**((ratings_list[i])-1))/math.log2(i+2))
    print(f"DCG_k = {summa}")
    return summa

def iDCG_k(ratings_list, k):
    print("iDCG_k")
    sorted_list = np.sort(ratings_list)[::-1]
    print(sorted_list)
    return DCG_k(sorted_list, k)

def NDCG_k(r, k):
    """
    ratings_list: np.array(n_items,)
    k: int
    """
    print(f"NDCG_k: {DCG_k(r, k) / iDCG_k(r, k)}")
    NDCG_k([5, 5, 4, 5, 2, 4, 5, 3, 5, 5, 2, 3, 0, 0, 1, 2, 2, 3, 0], 5)
```

```
DCG_k = 72.570354082694
iDCG_k
[5 5 5 5 5 5 4 4 3 3 3 2 2 2 2 1 0 0 0]
DCG_k = 91.40223268526115
NDCG_k: 0.7939669737892098
```

Рисунок 4.1 – Вывод консоли после оценки качества метрикой $NDCG$

Вывод

В ходе работы была реализована *ALS* факторизация, произведён поиск похожих фильмов и пользователей, а также написан код для реализации *NDCG* метрики и протестирован на примере.