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](#_Toc185016270)

[1 *ALS* факторизация 4](#_Toc185016271)

[2 Поиск похожих фильмов 9](#_Toc185016272)

[3 Поиск похожего пользователя 10](#_Toc185016273)

[4 Оценка качества метрикой *NDCG* 12](#_Toc185016274)

[Вывод 14](#_Toc185016275)

# 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-*1*m*.*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 воины 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 зип-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.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

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.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

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.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

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)

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Автоматически созданное описание

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.