Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра информатики

Лабораторная работа №9
Рекомендательные системы

Выполнил: Полевой Александр Вадимович
магистрант кафедры информатики
группа № 858641

Проверил: Стержанов Максим Валерьевич

Минск 2019

Набор данных ex9_movies.mat представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит две матрицы Y и R - рейтинг 1682 фильмов среди 943 пользователей. Значение Rij может быть равно 0 или 1 в зависимости от того оценил ли пользователь j фильм i. Матрица Y содержит числа от 1 до 5 - оценки в баллах пользователей, выставленные фильмам.

1. Загрузите данные ex9_movies.mat из файла.

In [1]:

```
import numpy as np
import scipy.io
from scipy import optimize

movie_data = scipy.io.loadmat('data/ex9_movies.mat')
Y = movie_data['Y']
R = movie_data['R']

Nm, Nu = Y.shape

print(f'Y.shape = {Y.shape}')
print(f'R.shape = {R.shape}')

print(f'Users: {Nu}, Movies: {Nm}')
```

```
Y.shape = (1682, 943)
R.shape = (1682, 943)
Users: 943, Movies: 1682
```

2. Выберите число признаков фильмов (n) для реализации алгоритма коллаборативной фильтрации.

In [2]:

```
x_feat_count = 4
```

3. Реализуйте функцию стоимости для алгоритма.

In [3]:

```
def h(theta, X):
    return np.dot(X, theta.T)
```

In [4]:

```
def cost(theta, X, y, r):
    y_pred = h(theta, X)
    y_pred = y_pred * r

    return np.sum(np.power((y_pred - y), 2))
```

In [5]:

```
def J_combined(theta, X, y, r, lmb = 0.):
    reg = 0
    error = cost(theta, X, y, r)

    if lmb != 0:
        reg += lmb * np.sum(np.square(theta))
        reg += lmb * np.sum(np.square(X))

    return (error + reg) / 2
```

4. Реализуйте функцию вычисления градиентов.

In [6]:

```
def unroll(theta, X):
    return np.concatenate([theta.flatten(), X.flatten()])

def roll_up(data, movies_c, users_c, feat_c):
    theta_end = users_c * feat_c
    theta = data[:theta_end].reshape(users_c, feat_c)
    X = data[theta_end:].reshape(movies_c, feat_c)

    return theta, X
```

In [7]:

```
def J_gd(data, Y, R, Nm, Nu, Nf, lmb = 0.):
    theta, X = roll_up(data, Nm, Nu, Nf)

    return J_combined(theta, X, Y, R, lmb)
```

In [8]:

```
def gd_step(data, Y, R, Nm, Nu, Nf, lmb = 0.):
    theta, X = roll_up(data, Nm, Nu, Nf)

    error = (h(theta, X) * R) - Y

    X_gd = np.dot(error, theta)
    theta_gd = np.dot(error.T, X)

    if lmb != 0:
        X_gd += lmb * X
        theta_gd += lmb * theta

    return unroll(theta_gd, X_gd)
```

5. При реализации используйте векторизацию для ускорения процесса обучения.

In [9]:

```
print('done')
```

```
done
```

6. Добавьте L2-регуляризацию в модель.

In [10]:

```
print('done')
```

```
done
```

7. Обучите модель с помощью градиентного спуска или других методов оптимизации.

In [11]:

```
def build_model(init_theta, init_X, Y, R, Nm, Nu, Nf, lmb = 0.):
    data = optimize.fmin_cg(
        J_gd,
        x0=unroll(init_theta, init_X),
        fprime=gd_step,
        args=(Y, R, Nm, Nu, Nf, lmb),
        maxiter=400,
        disp=True
    )

    return roll_up(data, Nm, Nu, Nf)
```

In [12]:

```
rand_theta = np.random.rand(Nu, x_feat_count)
rand_X = np.random.rand(Nm, x_feat_count)
calc_theta, calc_X = build_model(rand_theta, rand_X, Y, R, Nm, Nu, x_feat_count)
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 32802.722355
         Iterations: 400
         Function evaluations: 593
         Gradient evaluations: 593
```

In [ ]:

```

```

8. Добавьте несколько оценок фильмов от себя. Файл movie_ids.txt содержит индексы каждого из фильмов.

In [13]:

```
def load_vocab(filename):
    dict = {}
    with open(filename, encoding = "ISO-8859-1") as f:
        for line in f:
            (idx, name) = line.strip().split(' ', 1)
            dict[int(idx) -1] = name
    return dict
```

In [14]:

```
movies = load_vocab('data/movie_ids.txt')
```

9. Сделайте рекомендации для себя. Совпали ли они с реальностью?

In [15]:

```
def mse(y_true, y_pred):
    return np.sum((y_true - y_pred) ** 2) / y_true.size
```

In [16]:

```
def recommend_movies(predictions, R, user_id, top=10):
    predictions = predictions[:, user_id]
    not_watched_R = np.where(R[:, user_id] < 1)[0]
    predictions = predictions[not_watched_R]
    recommended_idx = np.argsort(predictions)
    recommended_idx = recommended_idx[::-1] # asc -> desc

    print(f'Top {top}:')
    for i in range(top):
        idx = recommended_idx[i]
        print(f'{predictions[idx]:.2f} - {movies[idx]}')
```

In [17]:

```
my_rank = np.zeros([Y.shape[0],1]).astype(int)

my_rank[0] = 3
my_rank[1] = 5
my_rank[49] = 3
my_rank[70] = 2
my_rank[71] = 3
my_rank[95] = 5
my_rank[203] = 3
my_rank[209] = 5
my_rank[221] = 3
my_rank[226] = 3
my_rank[227] = 3
my_rank[228] = 3
my_rank[229] = 3
my_rank[225] = 4
my_rank[404] = 5
my_rank[549] = 5
my_rank[406] = 3
my_rank[383] = 3
my_rank[454] = 5
my_rank[249] = 5

my_indexes = np.where(my_rank > 0)[0]

for idx in my_indexes:
    print(f"{movies[idx]}\t{my_rank[idx].item()}")
```

```
Toy Story (1995)        3
GoldenEye (1995)        5
Star Wars (1977)        3
Lion King, The (1994)   2
Mask, The (1994)        3
Terminator 2: Judgment Day (1991)       5
Back to the Future (1985)       3
Indiana Jones and the Last Crusade (1989)       5
Star Trek: First Contact (1996) 3
Die Hard 2 (1990)       4
Star Trek VI: The Undiscovered Country (1991)   3
Star Trek: The Wrath of Khan (1982)     3
Star Trek III: The Search for Spock (1984)      3
Star Trek IV: The Voyage Home (1986)    3
Fifth Element, The (1997)       5
Naked Gun 33 1/3: The Final Insult (1994)       3
Mission: Impossible (1996)      5
Spy Hard (1996) 3
Jackie Chan's First Strike (1996)       5
Die Hard: With a Vengeance (1995)       5
```

In [18]:

```
my_rated = (my_rank > 0).astype(int)
```

In [19]:

```
my_Y = np.c_[Y, my_rank]
my_Y.shape
```

Out[19]:

```
(1682, 944)
```

In [20]:

```
movies_mean = my_Y.mean(axis=1).reshape(-1, 1)
movies_mean.shape
```

Out[20]:

```
(1682, 1)
```

In [21]:

```
my_R = np.c_[R, my_rated]
my_R.shape
```

Out[21]:

```
(1682, 944)
```

In [22]:

```
my_Nm, my_Nu = my_Y.shape
my_feat_count = 100
```

In [23]:

```
init_theta = np.random.rand(my_Nu, my_feat_count)
init_X = np.random.rand(my_Nm, my_feat_count)
```

In [24]:

```
my_calc_theta, my_calc_X = build_model(init_theta, init_X, my_Y, my_R, my_Nm, my_Nu,
y_pred_gd = h(my_calc_theta, my_calc_X)
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 2464.351748
         Iterations: 400
         Function evaluations: 597
         Gradient evaluations: 597
```

In [25]:

```
print(f'mse = {mse(my_Y, y_pred_gd)}')
```

```
mse = 11.09484582726997
```

In [26]:

```
recommend_movies(y_pred_gd, my_R, 943, 10)
```

```
Top 10:
5.50 - Sliding Doors (1998)
5.24 - Empire Strikes Back, The (1980)
5.22 - Vertigo (1958)
5.18 - Free Willy (1993)
5.17 - Pink Floyd - The Wall (1982)
5.15 - Nightwatch (1997)
5.15 - Head Above Water (1996)
5.12 - Army of Darkness (1993)
5.06 - Secrets & Lies (1996)
5.03 - Right Stuff, The (1983)
```

10. Также обучите модель с помощью сингулярного разложения матриц. Отличаются ли полученные результаты?

In [27]:

```
def predict_using_svd(Y, feat_count):
    U, Sigma, Vt = np.linalg.svd(Y)

    U_f = U[:, : feat_count]
    S_f = np.diag(Sigma[: feat_count])
    Vt_f = Vt[: feat_count]

    predictions = np.dot(np.dot(U_f, S_f), Vt_f)

    return predictions
```

In [28]:

```
y_pred_svd = predict_using_svd(my_Y, 100)
print(f'mse = {mse(my_Y, y_pred_svd)}')
```

```
mse = 0.2461391493665448
```

In [29]:

```
recommend_movies(y_pred_svd, my_R, 943, 10)
```

```
Top 10:
1.47 - Grand Day Out, A (1992)
1.47 - Brazil (1985)
1.25 - Manon of the Spring (Manon des sources) (1986)
1.21 - Age of Innocence, The (1993)
1.09 - Monty Python and the Holy Grail (1974)
1.07 - Highlander (1986)
1.07 - Haunted World of Edward D. Wood Jr., The (1995)
1.01 - D3: The Mighty Ducks (1996)
1.01 - Black Sheep (1996)
1.00 - Kolya (1996)
```

Вывод

Рекомендательные системы — программы, которые пытаются предсказать, какие объекты (фильмы, музыка, книги, новости, веб-сайты) будут интересны пользователю, имея определенную информацию о его профиле.

Две основные стратегии создания рекомендательных систем — фильтрация на основе содержания и коллаборативная фильтрация. При фильтрации на основе содержания создаются профили пользователей и объектов, профили пользователей могут включать демографическую информацию или ответы на определённый набор вопросов, профили объектов могут включать названия жанров, имена актёров, имена исполнителей и другую атрибутивную информацию в зависимости от типа объекта. Например, в Music Genome Project музыкальный аналитик оценивает каждую композицию по сотням различных музыкальных характеристик, которые могут использоваться для выявления музыкальных предпочтений пользователя. При коллаборативной фильтрации используется информация о поведении пользователей в прошлом — например, информация о покупках или оценках. В этом случае не имеет значения, с какими типами объектов ведётся работа, но при этом могут учитываться неявные характеристики, которые сложно было бы учесть при создании профиля. Основная проблема этого типа рекомендательных систем — «холодный старт»: отсутствие данных о недавно появившихся в системе пользователях или объектах.

В процессе работы рекомендательные системы собирают данные о пользователях, используя сочетание явных и неявных методов.

Примеры явного сбора данных:

- запрос у пользователя оценки объекта по дифференцированной шкале;
- запрос у пользователя ранжировки группы объектов от наилучшего к наихудшему;
- предъявление пользователю двух объектов с вопросом о том, какой из них лучше;
- предложение создать список объектов, любимых пользователем.

Примеры неявного сбора данных:

- наблюдение за тем, что осматривает пользователь в интернет-магазинах или базах данных другого типа;
- ведение записей о поведении пользователя онлайн;
- отслеживание содержимого компьютера пользователя.