Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ЛАБОРАТОРНАЯ РАБОТА №9

«Рекомендательные системы»

Студент А. Ю. Омельчук

Преподаватель М. В. Стержанов

Минск 2019

ХОД РАБОТЫ

Задание.

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.
2. Выберите число признаков фильмов (n) для реализации алгоритма коллаборативной фильтрации.
3. Реализуйте функцию стоимости для алгоритма.
4. Реализуйте функцию вычисления градиентов.
5. При реализации используйте векторизацию для ускорения процесса обучения.
6. Добавьте L2-регуляризацию в модель.
7. Обучите модель с помощью градиентного спуска или других методов оптимизации.
8. Добавьте несколько оценок фильмов от себя. Файл movie_ids.txt содержит индексы каждого из фильмов.
9. Сделайте рекомендации для себя. Совпали ли они с реальностью?
10. Также обучите модель с помощью сингулярного разложения матриц. Отличаются ли полученные результаты?

Результат выполнения:

1. Код выгрузки данных из файла представлен ниже:

```
        file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex9_movies.mat')
        dataset = sio.loadmat(file_path)

        # Y is a 1682x943 matrix, containing ratings (1-5) of
        # 1682 movies on 943 users
        # R is a 1682x943 matrix, where R(i,j) = 1
        # if and only if user j gave a rating to movie i
        Y, R = dataset['Y'], dataset['R']
        X = np.zeros((1682, 10))  # 1682 X 10 matrix , num_movies X num_features matrix of movie features
        Theta = np.zeros((943, 10))  # 943 X 10 matrix, num_users X num_features matrix of user features
```

2. Код реализации:

```
num_users, num_movies, num_features = 4, 5, 3
```

3-6. Код реализации:

```
def cost_function(params, Y, R, num_users, num_movies, num_features, Lambda):
    """
    Returns the cost and gradient for the collaborative filtering problem
    """

    # Unfold the params
    X = params[:num_movies*num_features].reshape(num_movies,num_features)
    Theta = params[num_movies*num_features:].reshape(num_users,num_features)

    predictions = np.dot(X, Theta.T)
    err = (predictions - Y)
    J = 1/2 * np.sum((err**2) * R)

    # compute regularized cost function
    reg_X = Lambda/2 * np.sum(Theta**2)
    reg_Theta = Lambda/2 * np.sum(X**2)
    reg_J = J + reg_X + reg_Theta

    # Compute gradient
    X_grad = np.dot(err*R, Theta)
    Theta_grad = np.dot((err*R).T, X)
    grad = np.append(X_grad.flatten(), Theta_grad.flatten())

    # Compute regularized gradient
    reg_X_grad = X_grad + Lambda*X
    reg_Theta_grad = Theta_grad + Lambda*Theta
    reg_grad = np.append(reg_X_grad.flatten(), reg_Theta_grad.flatten())

    return J, grad, reg_J, reg_grad

X_test = X[:num_movies, :num_features]
Theta_test= Theta[:num_users, :num_features]
Y_test = Y[:num_movies, :num_users]
R_test = R[:num_movies, :num_users]
params = np.append(X_test.flatten(), Theta_test.flatten())

# Evaluate cost function
J, grad = cost_function(params, Y_test, R_test, num_users, num_movies, num_features,
0)[:2]
print("Cost at loaded parameters:", J)
J2, grad2 = cost_function(params, Y_test, R_test, num_users, num_movies, num_features,
1.5)[2:]
print("Cost at loaded parameters (lambda = 1.5):", J2)
```

Результат выполнения:

```
('Cost at loaded parameters:', 0.0)
('Cost at loaded parameters (lambda = 1.5):', 0.0)
```

7. Код реализации:

```
    Ynorm, Ymean = normalize_ratings(Y, R)
    num_users = Y.shape[1]
    num_movies = Y.shape[0]
    num_features = 10
    # Set initial Parameters (Theta,X)
    X = np.random.randn(num_movies, num_features)
    Theta = np.random.randn(num_users, num_features)
    initial_parameters = np.append(X.flatten(), Theta.flatten())
    Lambda = 10
    # Optimize parameters using Gradient Descent
    paramsFinal, J_history = gradient_descent(initial_parameters, Y, R, num_users, num_movies, num_features, 0.001, 400, Lambda)
```

Результат выполнения:

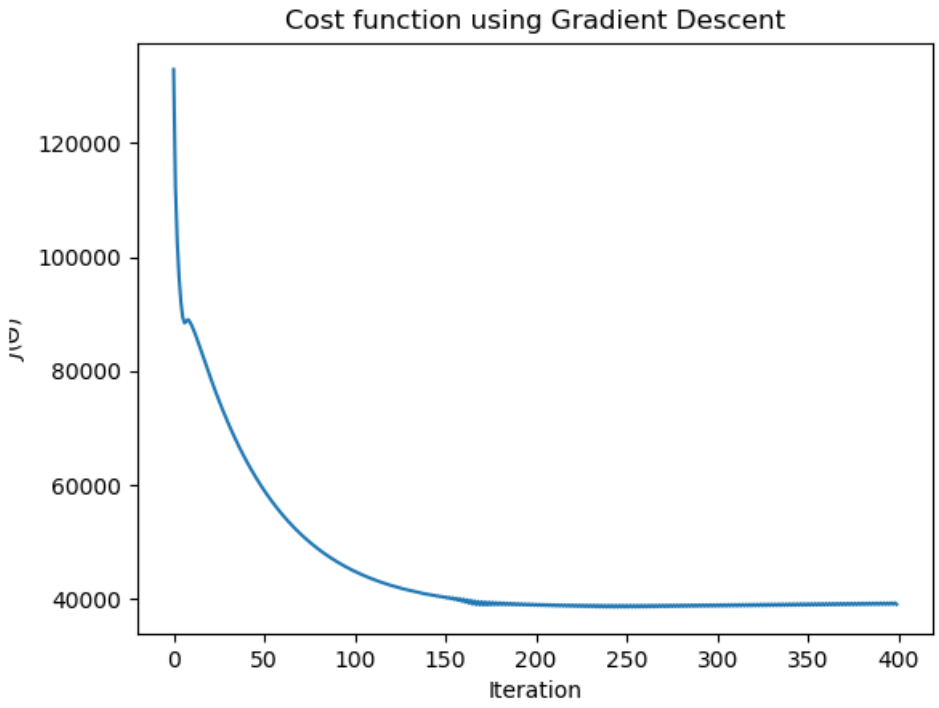

Рисунок 1 – график зависимости функции стоимости в зависимости от количества итераций

8. Код реализации:

```
def get_my_dataset():
    my_ratings = np.zeros((1682, 1))
    # Set my own estimation
    my_ratings[190] = 5
    my_ratings[63] = 5
    my_ratings[70] = 4
    my_ratings[68] = 5
    my_ratings[95] = 5
```

```
    return my_ratings
```

9. Код реализации:

```
print("Top recommendations for you:\n")

for i in range(10):
    print("Predicting rating", round(float(df[0][i]), 1), " for index", df[1][i])
```

Результат выполнения:

```
('Predicting rating', 9.3, ' for index', '100 Fargo (1996)')
('Predicting rating', 9.3, ' for index', '302 L.A. Confidential (1997)')
('Predicting rating', 9.3, ' for index', '285 Secrets & Lies (1996)')
('Predicting rating', 9.2, ' for index', '408 Close Shave, A (1995)')
('Predicting rating', 9.1, ' for index', '114 Wallace & Gromit: The Best of Aardman
Animation (1996)')
('Predicting rating', 9.0, ' for index', '172 Empire Strikes Back, The (1980)')
('Predicting rating', 8.9, ' for index', '169 Wrong Trousers, The (1993)')
('Predicting rating', 8.9, ' for index', '124 Lone Star (1996)')
('Predicting rating', 8.9, ' for index', '181 Return of the Jedi (1983)')
('Predicting rating', 8.9, ' for index', '23 Taxi Driver (1976)')
```

Результаты частично совпали. Т. к. все фильмы были сняты не позднее 2000 года, то я не смог сгенерировать хороший датасет, который бы описывал мои вкусы. Поэтому все фильмы, которые я нашёл в исходном датасете и которые я смотрел, в принципе, мне понравились, и я их оценил на 5. Из предложенных фильмов я смотрел только «Невероятные приключения Уоллеса и Громита: Неправильные штаны», который мне понравился. Поэтому можно сделать вывод, что фильтрация работает.

10. Код реализации:

```
Y = np.array(Y.T)
Y = np.append(Y, [get_my_dataset().flatten()], axis=0)
R = Y
user_ratings_mean = np.mean(R, axis=1)
R_demeaned = R - user_ratings_mean.reshape(-1, 1)
U, sigma, Vt = svds(R_demeaned, k=300)
sigma = np.diag(sigma)
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-
1, 1)

for i in all_user_predicted_ratings[943].argsort()[-15:][::-1]:
    print("Predicting rating", all_user_predicted_ratings[943][i], "index: ",
movieList[i])
```

Результат выполнения:

```
    ('Predicting rating', 0.7539285824950035, 'index: ', '462 Like Water For Chocolate
(Como agua para chocolate) (1992)')
    ('Predicting rating', 0.5298127098511799, 'index: ', '684 In the Line of Fire
(1993)')
    ('Predicting rating', 0.5216574724921866, 'index: ', '661 High Noon (1952)')
    ('Predicting rating', 0.490094540379511, 'index: ', '195 Terminator, The (1984)')
    ('Predicting rating', 0.48649407742898026, 'index: ', '521 Deer Hunter, The (1978)')
    ('Predicting rating', 0.47488213779727945, 'index: ', '95 Aladdin (1992)')
    ('Predicting rating', 0.46064422700289187, 'index: ', '762 Beautiful Girls (1996)')
    ('Predicting rating', 0.41264692486159277, 'index: ', '898 Postman, The (1997)')
    ('Predicting rating', 0.4039354465529767, 'index: ', '8 Babe (1995)')
    ('Predicting rating', 0.398605931501228, 'index: ', '519 Treasure of the Sierra
Madre, The (1948)')
```

Результаты очень сильно отличаются. Во втором случае просмотренных мной фильмов, которые мне понравились и которых не было в исходном датасете, больше, т. е. из этих топ-10 рекомендаций для меня я смотрел «Терминатор», «Аладин», «На линии огня». Сравнивая два сета рекоммендаций последний мне нравится больше.

Программный код:

```
    from goto import with_goto
    import scipy.io as sio
    import matplotlib.pyplot as plt
    import os
    import numpy as np
    import pandas as pd
    from scipy.sparse.linalg import svds


    def cost_function(params, Y, R, num_users, num_movies, num_features, Lambda):
        """
        Returns the cost and gradient for the collaborative filtering problem
        """

        # Unfold the params
        X = params[:num_movies*num_features].reshape(num_movies,num_features)
        Theta = params[num_movies*num_features:].reshape(num_users,num_features)

        predictions = np.dot(X, Theta.T)
        err = (predictions - Y)
        J = 1/2 * np.sum((err**2) * R)

        # compute regularized cost function
        reg_X =  Lambda/2 * np.sum(Theta**2)
        reg_Theta = Lambda/2 *np.sum(X**2)
        reg_J = J + reg_X + reg_Theta

        # Compute gradient
        X_grad = np.dot(err*R, Theta)
        Theta_grad = np.dot((err*R).T, X)
        grad = np.append(X_grad.flatten(), Theta_grad.flatten())

        # Compute regularized gradient
        reg_X_grad = X_grad + Lambda*X
        reg_Theta_grad = Theta_grad + Lambda*Theta
        reg_grad = np.append(reg_X_grad.flatten(), reg_Theta_grad.flatten())

        return J, grad, reg_J, reg_grad


    def normalize_ratings(Y, R):
        """
        normalized Y so that each movie has a rating of 0 on average, and returns the mean
rating in Ymean.
        """
        m, n = Y.shape[0], Y.shape[1]
        Ymean = np.zeros((m, 1))
        Ynorm = np.zeros((m, n))

        for i in range(m):
            Ymean[i] = np.sum(Y[i, :])/np.count_nonzero(R[i, :])
            Ynorm[i, R[i, :] == 1] = Y[i, R[i, :] == 1] - Ymean[i]

        return Ynorm, Ymean
```

```
    def
gradient_descent(initial_parameters,Y,R,num_users,num_movies,num_features,alpha,num_iters,Lamb
da):
        """
        Optimize X and Theta
        """
        # unfold the parameters
        X = initial_parameters[:num_movies*num_features].reshape(num_movies,num_features)
        Theta = initial_parameters[num_movies*num_features:].reshape(num_users,num_features)

        J_history = []

        for i in range(num_iters):
            params = np.append(X.flatten(), Theta.flatten())
            cost, grad = cost_function(params, Y, R, num_users, num_movies, num_features,
Lambda)[2:]

            # unfold grad
            X_grad = grad[:num_movies*num_features].reshape(num_movies,num_features)
            Theta_grad = grad[num_movies*num_features:].reshape(num_users,num_features)
            X = X - (alpha * X_grad)
            Theta = Theta - (alpha * Theta_grad)
            J_history.append(cost)

        paramsFinal = np.append(X.flatten(), Theta.flatten())
        return paramsFinal, J_history


    def get_my_dataset():
        my_ratings = np.zeros((1682, 1))
        # Set my own estimation
        my_ratings[190] = 5
        my_ratings[63] = 5
        my_ratings[70] = 4
        my_ratings[68] = 5
        my_ratings[95] = 5

        return my_ratings


    @with_goto
    def main():
        goto .task
        label .task
        # 1
        file_path = os.path.join(os.path.dirname(__file__), 'data', 'ex9_movies.mat')
        dataset = sio.loadmat(file_path)

        # Y is a 1682x943 matrix, containing ratings (1-5) of
        # 1682 movies on 943 users
        # R is a 1682x943 matrix, where R(i,j) = 1
        # if and only if user j gave a rating to movie i
        Y, R = dataset['Y'], dataset['R']
        X = np.zeros((1682, 10))  # 1682 X 10 matrix , num_movies X num_features matrix of movie
features
        Theta = np.zeros((943, 10))  # 943 X 10 matrix, num_users X num_features matrix of user
features

        # 2
```

```
        num_users, num_movies, num_features = 4, 5, 3

        # 3-6
        X_test = X[:num_movies, :num_features]
        Theta_test= Theta[:num_users, :num_features]
        Y_test = Y[:num_movies, :num_users]
        R_test = R[:num_movies, :num_users]
        params = np.append(X_test.flatten(), Theta_test.flatten())

        # Evaluate cost function
        J, grad = cost_function(params, Y_test, R_test, num_users, num_movies, num_features,
0)[:2]
        print("Cost at loaded parameters:", J)
        J2, grad2 = cost_function(params, Y_test, R_test, num_users, num_movies, num_features,
1.5)[2:]
        print("Cost at loaded parameters (lambda = 1.5):", J2)

        # 7
        # Normalize Ratings
        Ynorm, Ymean = normalize_ratings(Y, R)
        num_users = Y.shape[1]
        num_movies = Y.shape[0]
        num_features = 10
        # Set initial Parameters (Theta,X)
        X = np.random.randn(num_movies, num_features)
        Theta = np.random.randn(num_users, num_features)
        initial_parameters = np.append(X.flatten(), Theta.flatten())
        Lambda = 10
        # Optimize parameters using Gradient Descent
        paramsFinal,  J_history  =  gradient_descent(initial_parameters,  Y,  R,  num_users,
num_movies, num_features, 0.001, 400, Lambda)

        plt.plot(J_history)
        plt.xlabel("Iteration")
        plt.ylabel("$J(\Theta)$")
        plt.title("Cost function using Gradient Descent")

        plt.show()

        # 8
        # load movie list
        movieList  =  open(os.path.join(os.path.dirname(__file__),  'data',  'movie_ids.txt'),
"r").read().split("\n")[:-1]
        # Initialize my ratings
        my_ratings = get_my_dataset()

        print("My ratings:\n")
        for i in range(len(my_ratings)):
            if my_ratings[i] > 0:
                print("Rated", int(my_ratings[i]), "for index", movieList[i])

        # 9
        # unfold paramaters
        X = paramsFinal[:num_movies*num_features].reshape(num_movies, num_features)
        Theta = paramsFinal[num_movies*num_features:].reshape(num_users, num_features)
        # Predict rating
        p = np.dot(X, Theta.T)
        my_predictions = p[:, 0][:, np.newaxis] + Ymean
        df = pd.DataFrame(np.hstack((my_predictions, np.array(movieList)[:, np.newaxis])))
```

```
        df.sort_values(by=[0], ascending=False, inplace=True)
        df.reset_index(drop=True, inplace=True)
        print("Top recommendations for you:\n")

        for i in range(10):
            print("Predicting rating", round(float(df[0][i]), 1), " for index", df[1][i])

        # 10
        Y = np.array(Y.T)
        Y = np.append(Y, [get_my_dataset().flatten()], axis=0)
        R = Y
        user_ratings_mean = np.mean(R, axis=1)
        R_demeaned = R - user_ratings_mean.reshape(-1, 1)
        U, sigma, Vt = svds(R_demeaned, k=300)
        sigma = np.diag(sigma)
        all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-
1, 1)

        for i in all_user_predicted_ratings[943].argsort()[-15:][::-1]:
            print("Predicting   rating",   all_user_predicted_ratings[943][i],   "index:   ",
movieList[i])


    if __name__ == '__main__':
        main()
```