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 |  | К. Д. Зюсько |
| 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

1. 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:

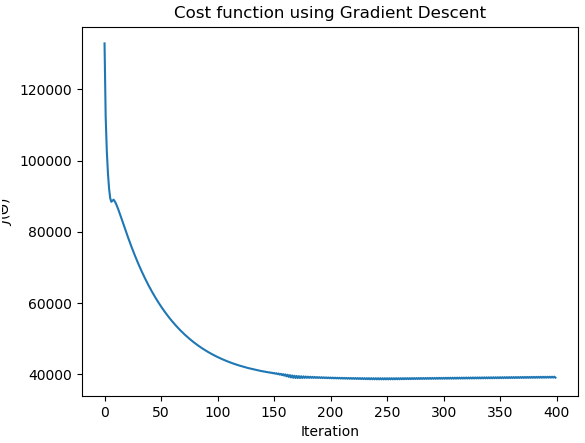


Рисунок 1 – график зависимости функции стоимости в зависимости от количества итераций

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 линни 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,Lambda):

"""

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()