The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Siyuan Xiao, Li Zhang, Shengyan Wen

Supervisor:
Qingyao Wu

Student ID：
201721045886, 201721045909, 201721045893

Grade:
Postgraduate

December 24, 2017

Recommender System Based on Matrix Decomposition

Abstract—Nowadays, recommender system is a popular application that applying data mining techniques. In this field, Matrix Decomposition is used frequently.

I. Introduction

This report will talk about the whole experiment I have made for a recommender system based on Matrix Factorization. Its content is organized as follow:

1) Section II contains the experiment steps.
2) Section III contains the code for the experiment.
3) Section IV makes conclusion for the experiment result.

II. Methods and Theory

Here we use the MovieLens-100k dataset which is placed in the requirement. The u1 database is used and it is already splitted into training set and validation set by the official.

Then the experiment will be performed by the following steps:

1) Load the data text into memory and do the transformation to get the specific column, and the cells that lack of data is set to 0 (or the average value).
2) For ALS method, we randomly initiate the Q matrix.
3) After that, we calculate P and Q alternatively until the iteration number is arrived or the RMSE has little change.
4) For SGD method, we randomly initiate the two matrices.
5) In each iteration, we randomly pick one row in P and Q, respectively.
6) We use these two selected rows to generate a prediction, then calculate the error and gradient for P and Q, respectively.
7) Similarly, we iterate until it converges or reaches the given number of iterations
8) Regardless of which method we use, do the whole prediction and calculate RMSE for the valid validation data.

III. Experiment

Here I placed the critical code for the experiment:

1) Preprocessing:

```
# created by Swain, 2017-12-19, 11:29

import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plot
from numpy import random

#load u1 from MovieLens-100k dataset
n_user = 943
n_item = 1682
trainpath = './u1.base'
testpath = './u1.test'

data_train = np.loadtxt(trainpath)
data_test = np.loadtxt(testpath)

def make_R(data):
    count = 0
    sum = 0
    R = np.zeros([n_user, n_item])
    for i in range(0, data.shape[0]):
        R[int(data[i, 0]) - 1, int(data[i, 1]) - 1] = int(data[i, 2])
        count = count + 1
        sum = sum + int(data[i, 2])

    avg = sum / count
    '''for i in range(0, R_train.shape[0]):
        for j in range(0, R_train.shape[1]):
            if R_train[i, j] == 0:
                R_train[i, j] = avg'''
    return R

R_train = make_R(data_train)
R_vali = make_R(data_train)

Np = np.sum(R_train, axis=1)
Nq = np.sum(R_train, axis=0)

random.seed(24)
```

2) ALS:

```
#ALS
k = 10
lambda_ = 0.03
iteration = 1000

P = random.random([n_user, k]) * 5
Q = random.random([n_item, k]) * 5

Lambda = lambda_ * np.eye(k)

R = R_train

for n in range(0, iteration):
    print ("start the " + str(n + 1) + "-th iteration")

    #calculate P matrix
    for i in range(0, n_user):
        P[i] = np.dot(inv(np.dot(Q.T, Q) + Lambda), np.dot(Q.T, R[i]))

    #calculate Q matrix
    for i in range(0, n_item):
        Q[i] = np.dot(inv(np.dot(P.T, P) + Lambda), np.dot(P.T, R[:, i]))

    #do prediction
    R_predict = np.dot(P, Q.T)

    #calculate error
    RMSE = 0
    count = 0
    for i in range(0, n_user):
        for j in range(0, n_item):
            if R_vali[i, j] != 0:
                RMSE = RMSE + np.power(R_vali[i, j] - R_predict[i, j], 2)
                count = count + 1
    RMSE = np.sqrt(RMSE / count)
    print ("  RMSE = " + str(RMSE))


    if n > 0 and np.abs(RMSE - last_RMSE) < 0.0001:
        break;
    last_RMSE = RMSE
```

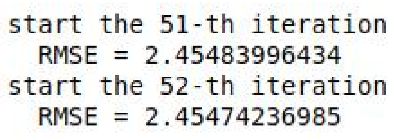
3) SGD:

```
#SGD
k = 10
lr = 0.05
lambda_p = 0.03
lambda_q = 0.03
iteration = 1000000

#P = np.ones([n_user, k])
#Q = np.ones([n_item, k])
P = random.random([n_user, k])
Q = random.random([n_item, k])

R = R_train

for n in range(0, iteration):
    #randomly pick a row from P and Q
    u = random.randint(0, n_user - 1)
    i = random.randint(0, n_item - 1)

    #calculate error and gradient
    error = R[u, i] - np.dot(P[u], Q.T[:, i])
    deriv_p = lambda_p * P[u] - error * Q[i]
    deriv_q = lambda_q * Q[i] - error * P[u]

    #do updation
    P[u] = P[u] - lr * deriv_p
    Q[i] = Q[i] - lr * deriv_q

    print (str(n + 1) + "-th iteration: error = " + str(error))

#do prediction and calculate error
R_predict = np.dot(P, Q.T)
RMSE = 0
count = 0
for i in range(0, n_user):
    for j in range(0, n_item):
        if R_vali[i, j] != 0:
            RMSE = RMSE + np.power(R_vali[i, j] - R_predict[i, j], 2)
            count = count + 1
RMSE = np.sqrt(RMSE / count)
print ("  RMSE = " + str(RMSE))
```

IV. CONCLUSION

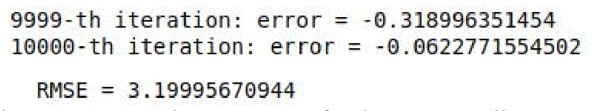
For ALS method, it ends with the 52-nd iteration. For SGD method, we do 100,000 iterations. Here is the experiment result gained:

1) For ALS:



```
start the 51-th iteration
  RMSE = 2.45483996434
start the 52-th iteration
  RMSE = 2.45474236985
```

2) For SGD:



```
9999-th iteration: error = -0.318996351454
10000-th iteration: error = -0.0622771554502

  RMSE = 3.19995670944
```

Then we can draw a conclusion according to the experiment:

1) ALS converges with far less iterations but may be likely to have overfitting problem.
2) SGD should run much more iterations since the data used in each iteration just generate one prediction.
3) If we set the undetermined cells of the training dataset to the average value (while the data shown is in case that setting to 0) of all training data, the RMSE of prediction can be reduced significantly.