South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

December 25, 2017

Author:
Shengjie Wang and Rui Lv and
Zhixiang Xue
王圣杰　吕睿　薛志翔

Supervisor:
Qingyao Wu

Student ID：
201530612873 and 20153061247
7 and 201530613313

Grade:
Undergraduate

Recommender System Based on Matrix Decomposition

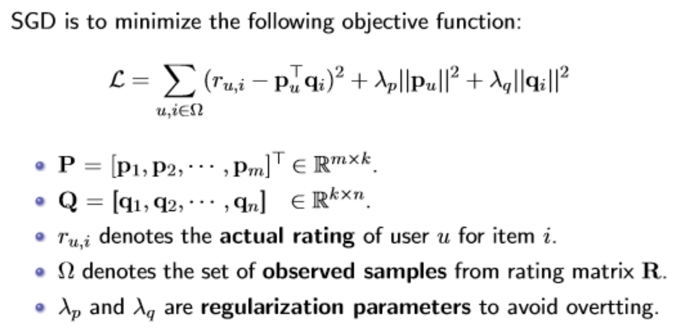
Abstract—This experiment focus on building a Recommender System Based on Matrix Decomposition.

I. Introduction

This experiment is for:

1. Explore the construction of recommended system.
2. Understand the principle of matrix decomposition.
3. Be familiar to the use of gradient descent.
4. Construct a recommendation system under small-scale dataset, cultivate engineering ability.

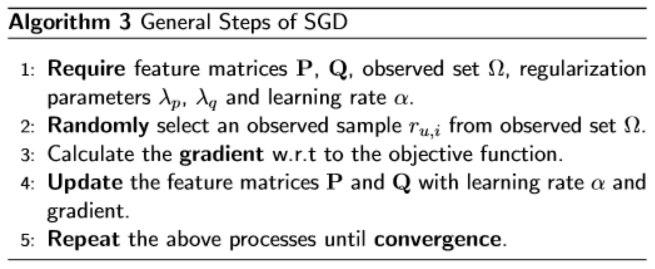
II. Methods and Theory



SGD is to minimize the following objective function:

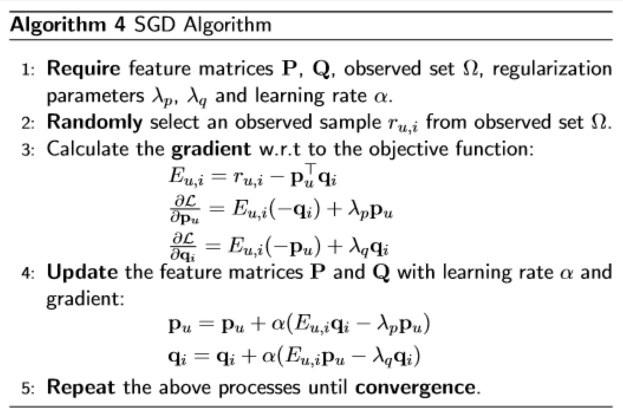
$$\mathcal{L} = \sum_{u,i\in\Omega} (r_{u,i} - \mathbf{p}_u^\top \mathbf{q}_i)^2 + \lambda_p ||\mathbf{p}_u||^2 + \lambda_q ||\mathbf{q}_i||^2$$

- $\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_m]^\top \in \mathbb{R}^{m\times k}$.
- $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n] \quad \in \mathbb{R}^{k\times n}$.
- $r_{u,i}$ denotes the **actual rating** of user u for item i.
- Ω denotes the set of **observed samples** from rating matrix $\mathbf{R}$.
- λ_p and λ_q are **regularization parameters** to avoid overtting.



Algorithm 3 General Steps of SGD

```
1: Require feature matrices $\mathbf{P}$, $\mathbf{Q}$, observed set $\Omega$, regularization
   parameters $\lambda_p$, $\lambda_q$ and learning rate $\alpha$.
2: Randomly select an observed sample $r_{u,i}$ from observed set $\Omega$.
3: Calculate the gradient w.r.t to the objective function.
4: Update the feature matrices $\mathbf{P}$ and $\mathbf{Q}$ with learning rate $\alpha$ and
   gradient.
5: Repeat the above processes until convergence.
```



Algorithm 4 SGD Algorithm

```
1: Require feature matrices $\mathbf{P}$, $\mathbf{Q}$, observed set $\Omega$, regularization
   parameters $\lambda_p$, $\lambda_q$ and learning rate $\alpha$.
2: Randomly select an observed sample $r_{u,i}$ from observed set $\Omega$.
3: Calculate the gradient w.r.t to the objective function:
       $E_{u,i} = r_{u,i} - \mathbf{p}_u^\top \mathbf{q}_i$
       $\frac{\partial \mathcal{L}}{\partial \mathbf{p}_u} = E_{u,i}(-\mathbf{q}_i) + \lambda_p \mathbf{p}_u$
       $\frac{\partial \mathcal{L}}{\partial \mathbf{q}_i} = E_{u,i}(-\mathbf{p}_u) + \lambda_q \mathbf{q}_i$
4: Update the feature matrices $\mathbf{P}$ and $\mathbf{Q}$ with learning rate $\alpha$ and
   gradient:
       $\mathbf{p}_u = \mathbf{p}_u + \alpha(E_{u,i}\mathbf{q}_i - \lambda_p \mathbf{p}_u)$
       $\mathbf{q}_i = \mathbf{q}_i + \alpha(E_{u,i}\mathbf{p}_u - \lambda_q \mathbf{q}_i)$
5: Repeat the above processes until convergence.
```

III. Experiment

A. *Dataset*

1. Utilizing MovieLens-100k dataset.
2. u.data -- Consisting 10,000 comments from 943 users out of 1682 movies. At least, each user comment 20 videos. Users and movies are numbered consecutively from number 1 respectively. The data is sorted randomly
3. u1.base / u1.test are train set and validation set respectively, seperated from dataset u.data with proportion of 80% and 20%. It also make sense to train set and validation set from u1.base / u1.test to u5.base / u5.test.
4. You can also construct train set and validation set according to your own evaluation method.

B. *Experiment Step*

Using alternate least squares optimization(ALS):

1. Read the data set and divide it (or use u1.base / u1.test to u5.base / u5.test directly). Populate the original scoring matrix against the raw data, and fill 0 for null values.
2. Initialize the user factor matrix and the item (movie) factor matrix , where is the number of potential features.
3. Determine the loss function and the hyperparameter learning rate and the penalty factor .

4. Use alternate least squares optimization method to decompose the sparse user score matrix, get the user factor matrix and item (movie) factor matrix:

   4.1.With fixd item factor matrix, find the loss partial derivative of each row (column) of the user factor matrices, ask the partial derivative to be zero and update the user factor matrices.

   4.2 With fixd user factor matrix, find the loss partial derivative of each row (column) of the item factor matrices, ask the partial derivative to be zero and update the item

   4.3 Calculate the on the validation set, comparing with the of the previous iteration to determine if it has converged.

5. Repeat step 4. several times, get a satisfactory user factor matrix and an item factor matrix , Draw a curve with varying iterations.

6. The final score prediction matrix is obtained by multiplying the user factor matrix and the transpose of the item factor matrix .

Using stochastic gradient descent method(SGD):

1. Read the data set and divide it (or use u1.base / u1.test to u5.base / u5.test directly). Populate the original scoring matrix against the raw data, and fill 0 for null values.
2. Initialize the user factor matrix and the item (movie) factor matrix , where is the number of potential features.
3. Determine the loss function and hyperparameter learning rate and the penalty factor .
4. Use the stochastic gradient descent method to decompose the sparse user score matrix, get the user factor matrix and item (movie) factor matrix:
   4.1 Select a sample from scoring matrix randomly;
   4.2 Calculate this sample's loss gradient of specific row(column) of user factor matrix and item factor matrix;
   4.3 Use SGD to update the specific row(column) of and ;
   4.4 Calculate the on the validation set, comparing with the of the previous iteration to determine if it has converged.
5. Repeat step 4. several times, get a satisfactory user factor matrix and an item factor matrix , **Draw a curve with varying iterations**.
6. The final score prediction matrix is obtained by multiplying the user factor matrix and the transpose of the item factor matrix .

C. Implementation

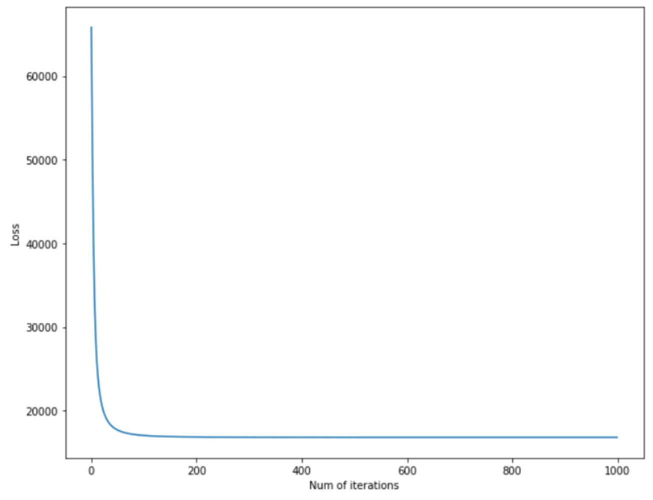
Source code:

train.py

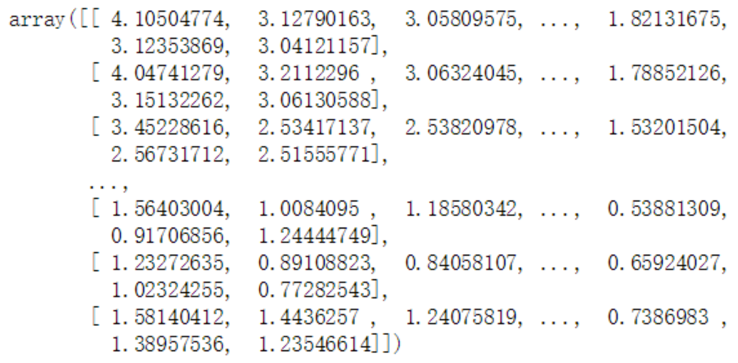
```
#从文件中读入训练集和测试集数据
import numpy as np
X_train = np.loadtxt('u1.base')
X_train = X_train[:,0:3]
X_test = np.loadtxt('u1.test')
X_test = X_test[:,0:3]

#根据训练集数据填充原始评分矩阵 R_train
R_train=np.zeros((943,1682))
for i in range(8000):
    uid=X_train[i,0]
    iid=X_train[i,1]
    rating=X_train[i,2]
    R_train[int(uid-1),int(iid-1)]=rating
#根据测试集数据填充原始评分矩阵 R_test
R_test=np.zeros((943,1682))
for i in range(8000):
    uid=X_test[i,0]
    iid=X_test[i,1]
    rating=X_test[i,2]
    R_test[int(uid-1),int(iid-1)]=rating

#定义 loss 函数
def loss(R,P,Q,lamda=0.02):
    e=0
    for i in range(len(R)):
        for j in range(len(R[i])):
            if R[i][j] > 0:
                e = e + pow(R[i][j] - np.dot(P[i,:],Q[:,j]), 2)
                for k in range(K):
                  e = e + lamda * (pow(P[i][k],2) + pow(Q[k][j],2))
        return e

#定义训练函数
def matrix_factorization(R, R_t, P, Q, K, steps=1000,
alpha=0.001, lamda=0.02, slowRate = 0.99):
    Loss=np.zeros((steps,1)) #初始化 Loss
    num_it=0 #最终循环次数
    preRmse = 10000000.0 #初始化一个大一点的数
    for step in range(steps):
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - np.dot(P[i,:],Q[:,j])
                    for k in range(K):
                        P[i][k] = P[i][k] + alpha * (eij * Q[k][j] -
lamda * P[i][k])
                        Q[k][j] = Q[k][j] + alpha * (eij * P[i][k] -
lamda * Q[k][j])
        nowRmse=loss(R_t,P,Q)
#计算根据本次循环得到的 P，Q 在测试集上的 loss
        if nowRmse<preRmse:
```

```
  #如果本次循环得到的 loss 小于之前的 loss，就更新之前的 loss 值
          preRmse = nowRmse
        elif nowRmse-preRmse<=0.01:
  #如果本次循环得到的 loss 比之前的 loss 增加量小于 0.01，则记录最终循环次数为本次循环，跳出循环
          num_it=step
          break
        #在前面两种情况之外的，即本次的 loss 比之前 loss 大了超过 0.01，意味着 alpha 过大，所以用 alpha*=slowRate 来减小 alpha
        Loss[step]=nowRmse
        alpha*=slowRate
        num_it=steps

  return P, Q, Loss, num_it

  K=3
  P = np.random.rand(943,3)
  Q = np.random.rand(3,1682)
  # 将 X_train，X_test，随机初始化的 P 和 Q，还有 K 传入训练函数进行训练
nP,nQ,L,n_it=matrix_factorization(R_train,R_test,P,Q,K)

  #绘制 Loss 随迭代次数变化的曲线图
  import matplotlib.pyplot as plt
  x=np.arange(0,n_it,1)
  plt.rcParams['figure.figsize'] = (10.0, 8.0)
  plt.plot(x,L)
  plt.xlabel('Num of iterations')
  plt.ylabel('Loss')
  plt.show()

  # 根据训练得到的 nP,nQ 计算最终的评分预测矩阵 R_pre
  R_pre=nP.dot(nQ)
  R_pre
```

Loss 随迭代次数变化的曲线图



根据训练得到的 nP,nQ 计算最终的评分预测矩阵 R_pre



```
array([[ 4.10504774,  3.12790163,  3.05809575, ...,  1.82131675,
         3.12353869,  3.04121157],
       [ 4.04741279,  3.2112296 ,  3.06324045, ...,  1.78852126,
         3.15132262,  3.06130588],
       [ 3.45228616,  2.53417137,  2.53820978, ...,  1.53201504,
         2.56731712,  2.51555771],
       ...,
       [ 1.56403004,  1.0084095 ,  1.18580342, ...,  0.53881309,
         0.91706856,  1.24444749],
       [ 1.23272635,  0.89108823,  0.84058107, ...,  0.65924027,
         1.02324255,  0.77282543],
       [ 1.58140412,  1.4436257 ,  1.24075819, ...,  0.7386983 ,
         1.38957536,  1.23546614]])
```

IV. CONCLUSION

We used stochastic gradient descent method to complete the experiment. After adjusting the parameters, the loss converge with iterations.
During this experiment, we explored the construction of recommended system. We understood the principle of matrix decomposition. And, we understood stochastic gradient descent method more. It is a good exercise to understand Machine Learning for us.