

导航

博客园

首 页

新随笔

联 系

订 阅

管 理

<2018年11月>

日

一

二

三

四

五

六

28

29

30

31

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

1

2

3

4

5

6

7

8

公告

昵称：1357

园龄：2年4个月

粉丝：15

关注：0

+加关注

搜索

找找看

谷歌搜索

常用链接

我的随笔

我的评论

我的参与

最新评论

我的标签

我的标签

机器学习(45)

工具学习(33)

图像处理(13)

深度学习(7)

矩阵运算(6)

论文学习(1)

随笔档案

2018年9月 (8)

2018年8月 (7)

2018年2月 (5)

2017年2月 (11)

2017年1月 (15)

2016年12月 (22)

2016年11月 (13)

2016年10月 (2)

2016年9月 (2)

2016年8月 (5)

2016年7月 (10)

最新评论

1. Re:Theano 学习四pool\_2d@larissa\_liu忽略边缘，比如5\*5做2\*2的下采样，忽略之后相当于4\*4的输入做下采样，输出2\*2，不忽略则输出3\*3。...--1357

2. Re:Theano 学习四pool\_2d没有看懂当ignore\_border=True时，到底怎么运算--larissa\_liu

阅读排行榜

## 非对称SVD电影推荐系统

采用1M MovieLensz数据(80%train, 20%test, UserIDs range between 1 and 6040 ,MovieIDs range between 1 and 3952, From <http://files.grouplens.org/datasets/movielens/>)

进行训练和测试，在k仅为10时，得到最佳RMSE为0.854743。在100k数据上k=100时最佳RMSE为0.916602。

以下公式和文字来自[陈靖\\_](http://blog.csdn.net/zhongkejingwang/article/details/43083603)的博文 <http://blog.csdn.net/zhongkejingwang/article/details/43083603>，包含详尽的描述和推导。

参考论文为《[Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model Yehuda Koren](#)》

## SVD

通过对A矩阵进行奇异值分解：

$$A = U \Sigma V^T$$

V是 $n \times n$ 的正交阵，U是 $m \times m$ 的正交阵，Σ是 $m \times n$ 的对角阵

现在可以来对A矩阵的映射过程进行分析了：如果在n维空间中找到一个（超）矩形，其边都落在A'A的特征向量的方向上，那么经过A变换后的形状仍然为（超）矩形！

vi为A'A的特征向量，称为A的右奇异向量，ui=Avi实际上为AA'的特征向量，称为A的左奇异向量。下面利用SVD证明文章一开始的满秩分解：

$$A = \left[ \begin{array}{ccc|ccc} u_1 & \cdots & u_k & u_{k+1} & \cdots & u_m \end{array} \right] \left[ \begin{array}{ccc|ccc} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_k & & & \\ \hline & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{array} \right] \left[ \begin{array}{c} v_1^T \\ \vdots \\ v_k^T \\ \hline v_{k+1}^T \\ \vdots \\ v_n^T \end{array} \right]$$

利用矩阵分块乘法展开得：

$$A = \left[ \begin{array}{ccc} u_1 & \cdots & u_k \end{array} \right] \left[ \begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{array} \right] \left[ \begin{array}{c} v_1^T \\ \vdots \\ v_k^T \end{array} \right] + \left[ \begin{array}{ccc} u_{k+1} & \cdots & u_m \end{array} \right] \left[ \begin{array}{ccc} 0 & & \\ & \ddots & \\ & & 0 \end{array} \right] \left[ \begin{array}{c} v_{k+1}^T \\ \vdots \\ v_n^T \end{array} \right]$$

可以看到第二项为0，有

$$A = \left[ \begin{array}{ccc} u_1 & \cdots & u_k \end{array} \right] \left[ \begin{array}{ccc} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{array} \right] \left[ \begin{array}{c} v_1^T \\ \vdots \\ v_k^T \end{array} \right]$$

令

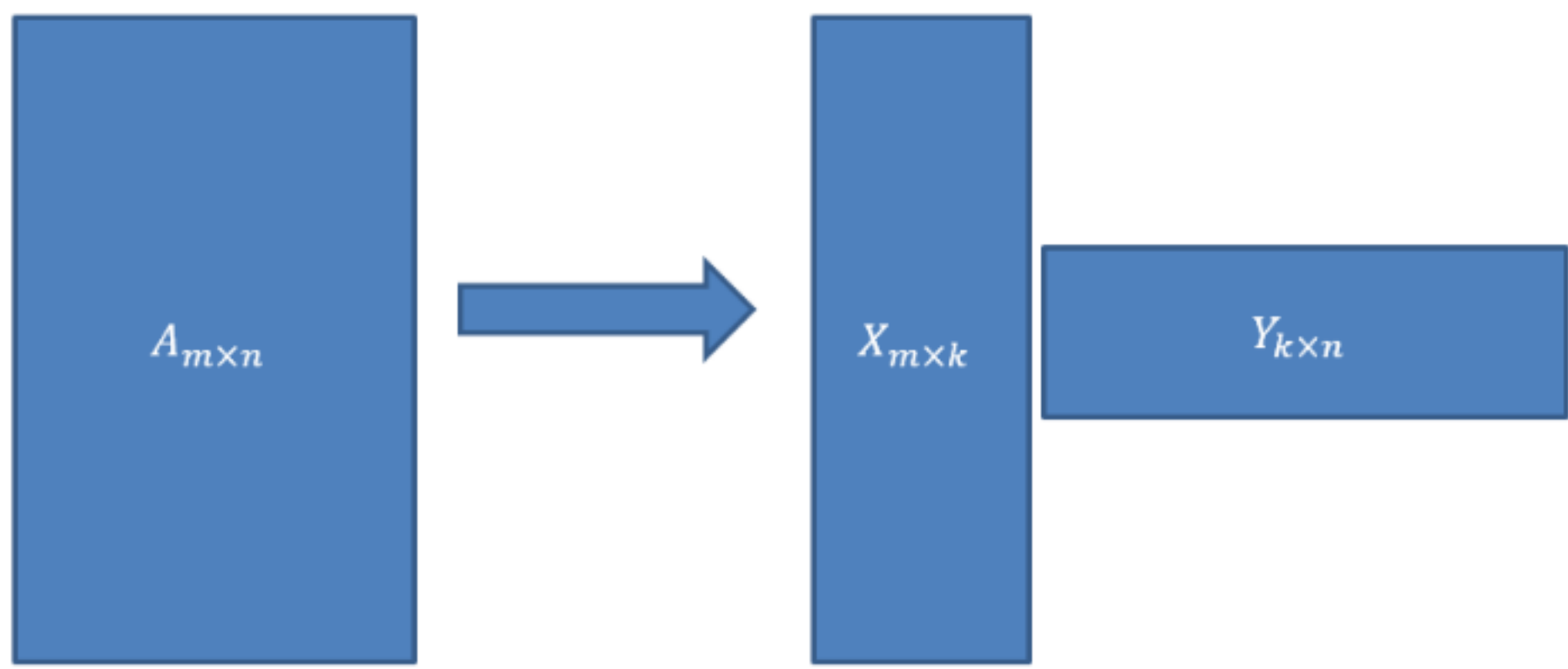
1. 神经网络回顾-Relu激活函数(27980)
2. 将数据集做成VOC2007格式用于Faster-RCNN训练(8708)
3. 卷积层和池化层(7133)
4. 矩阵的QR分解（三种方法）Python实现(5417)
5. BP神经网络——交叉熵作代价函数(2416)
- 评论排行榜
1. Theano 学习四pool\_2d(2)
- 推荐排行榜
1. 卷积层和池化层(1)
2. 将数据集做成VOC2007格式用于Faster-RCNN训练(1)
3. 图像处理3Felzenszwalb算法的Python实现(1)
4. 神经网络回顾-Relu激活函数(1)

$$X = \begin{bmatrix} u_1 & \cdots & u_k \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix} = \begin{bmatrix} \sigma_1 u_1 & \cdots & \sigma_k u_k \end{bmatrix}$$

$$Y = \begin{bmatrix} v_1^T \\ \vdots \\ v_k^T \end{bmatrix}$$

则A=XY即是A的满秩分解。

对任意一个矩阵A，都有它的满秩分解(k=Rank(A))：



那么刚才的评分矩阵R也存在这样一个分解，所以可以用两个矩阵P和Q的乘积来表示评分矩阵R：

$$R_{U \times I} = P_{U \times K} Q_{K \times I}$$

上图中的U表示用户数，I表示商品数。然后就是利用R中的已知评分训练P和Q使得P和Q相乘的结果最好地拟合已知的评分，那么未知的评分也就可以用P的某一行乘上Q的某一列得到了：

$$\hat{r}_{ui} = p_u^T q_i$$

这是预测用户u对商品i的评分，它等于P矩阵的第u行乘上Q矩阵的第i列。这个是最基本的SVD算法，那么如何通过已知评分训练得到P和Q的具体数值呢？

假设已知的评分为：

$$r_{ui}$$

则真实值与预测值的误差为：

$$e_{ui} = r_{ui} - \hat{r}_{ui}$$

继而可以计算出总的误差平方和：

$$SSE = \sum_{u,i} {e_{ui}}^2 = \sum_{u,i} \left( r_{ui} - \sum_{k=1}^K p_{uk} q_{ki} \right)^2$$

### Basic SVD

利用梯度下降法可以求得SSE在Puk变量（也就是P矩阵的第u行第k列的值）处的梯度：

$$\frac{\partial}{\partial p_{uk}}SSE=\frac{\partial}{\partial p_{uk}}\left(e_{ui}^2\right)$$

利用求导链式法则，e^2先对e求导再乘以e对Puk的求导：

$$\frac{\partial}{\partial p_{uk}}\left(e_{ui}^2\right)=2e_{ui}\frac{\partial}{\partial p_{uk}}e_{ui}$$

由于

$$e_{ui}=r_{ui}-\hat{r}_{ui}=r_{ui}-p_u^Tq_i=r_{ui}-\sum_{k=1}^Kp_{uk}q_{ki}$$

所以

$$\frac{\partial}{\partial p_{uk}}e_{ui}=\frac{\partial}{\partial p_{uk}}\left(r_{ui}-\sum_{k=1}^Kp_{uk}q_{ki}\right)$$

上式中括号里的那一坨式子如果展开来看的话，其与Puk有关的项只有PukQki，其他的无关项对Puk的求导均等于0

所以求导结果为：

$$\frac{\partial}{\partial p_{uk}}e_{ui}=\frac{\partial}{\partial p_{uk}}\left(r_{ui}-\sum_{k=1}^Kp_{uk}q_{ki}\right)=-q_{ki}$$

所以

$$\frac{\partial}{\partial p_{uk}}SSE=\frac{\partial}{\partial p_{uk}}\left(e_{ui}^2\right)=2e_{ui}\frac{\partial}{\partial p_{uk}}e_{ui}=-2e_{ui}q_{ki}$$

为了让式子更简洁，令

$$SSE=\frac{1}{2}\sum_{u,i}e_{ui}^2=\frac{1}{2}\sum_{u,i}\left(r_{ui}-\sum_{k=1}^Kp_{uk}q_{ki}\right)^2$$

这样做对结果没有影响，只是为了把求导结果前的2去掉，更好看点。得到

$$\frac{\partial}{\partial p_{uk}}SSE=-e_{ui}q_{ki}$$

现在得到了目标函数在Puk处的梯度了，那么按照梯度下降法，将Puk往负梯度方向变化：

令更新的步长（也就是学习速率）为

$$\eta$$

则Puk的更新式为

$$p_{uk}':=p_{uk}-\eta(-e_{ui}q_{ki}):=p_{uk}+\eta e_{ui}q_{ki}$$

同样的方式可得到Qik的更新式为

$$q_{ki}':=q_{ki}-\eta(-e_{ui}p_{uk}):=q_{ki}+\eta e_{ui}p_{uk}$$

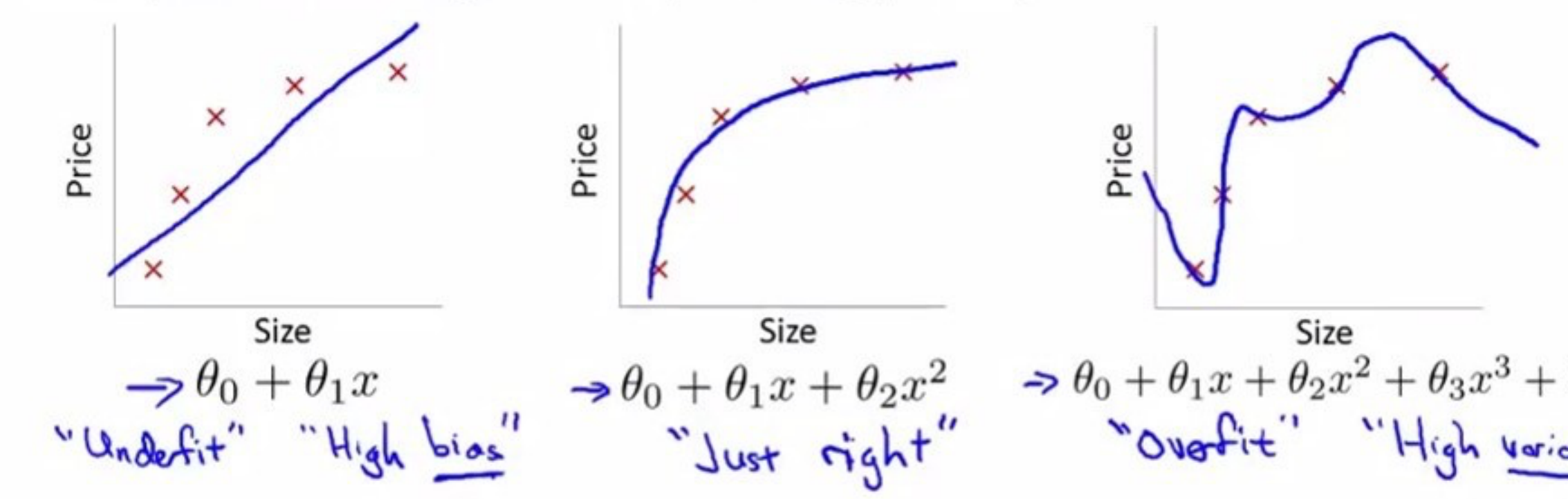
得到了更新的式子，现在开始来讨论这个更新要怎么进行。有两种选择：1、计算完所有已知评分的预测误差后再对P、Q进行更新。2、每计算完一个eui后立即对Pu和qi进行更新。这两种方式都有名称，分别叫：1、批梯度下降。2、随机梯度下降。两者的区别就是批梯度下降在下一轮迭代才能使用本次迭代的更新值，随机梯度下降本次迭代中当前样本使用的值可能就是上一个样本更新的值。由于随机性可以带来很多好处，比如有利于避免局部最优解，所以现在大多倾向于使用随机梯度下降进行更新。



RSVD

上面就是基本的SVD算法，但是，问题来了，上面的训练是针对已知评分数据的，过分地拟合这部分数据有可能导致模型的测试效果很差，在测试集上面表现很糟糕。这就是过拟合问题，关于过拟合与欠拟合可以看一下这张图

Example: Linear regression (housing prices)



**Overfitting:** If we have too many features, the learned hypothesis may fit the training set very well ( $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \approx 0$ ), but fails to generalize to new examples (predict prices on new examples).

第一个是欠拟合，第二个刚好，第三个过拟合。那么如何避免过拟合呢？那就是在目标函数中加入正则化参数（加入惩罚项），对于目标函数来说，P矩阵和Q矩阵中的所有值都是变量，这些变量在不知道哪个变量会带来过拟合的情况下，对所有变量都进行惩罚：

$$SSE = \frac{1}{2} \sum_{u,i} e_{ui}^2 + \frac{1}{2} \lambda \sum_u |p_u|^2 + \frac{1}{2} \lambda \sum_i |q_i|^2$$

这时候目标函数对Puk的导数就发生了变化了，现在就来求加入惩罚项后的导数。

$$\begin{aligned} SSE &= \frac{1}{2} \sum_{u,i} e_{ui}^2 + \frac{1}{2} \lambda \sum_u |p_u|^2 + \frac{1}{2} \lambda \sum_i |q_i|^2 \\ &= \frac{1}{2} \sum_{u,i} e_{ui}^2 + \frac{1}{2} \lambda \sum_u \sum_{k=0}^K p_{uk}^2 + \frac{1}{2} \lambda \sum_i \sum_{k=0}^K q_{ki}^2 \end{aligned}$$

括号里第一项对Puk的求导前面已经求过了，第二项对Puk的求导很容易求得，第三项与Puk无关，导数为0，所以

$$\frac{\partial}{\partial p_{uk}} SSE = -e_{ui} q_{ki} + \lambda p_{uk}$$

同理可得SSE对qik的导数为

$$\frac{\partial}{\partial q_{ik}} SSE = -e_{ui} p_{uk} + \lambda q_{ik}$$

将这两个变量往负梯度方向变化，则更新式为

$$\begin{aligned} p_{uk} &\leftarrow p_{uk} + \eta (e_{ui} q_{ki} - \lambda p_{uk}) \\ q_{ki} &\leftarrow q_{ki} + \eta (e_{ui} p_{uk} - \lambda q_{ik}) \end{aligned}$$

这就是正则化后的SVD，也叫RSVD。

加入偏置的SVD、RSVD

关于SVD算法的变种太多了，叫法也不统一，在预测式子上加点参数又会出来一个名称。由于用户对商品的打分不仅取决于用户和商品间的某种关系，还取决于用户和商品独有的性质，Koren将SVD的预测公式改成这样

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i$$

第一项为总的平均分，bu为用户u的属性值，bi为商品i的属性值，加入的这两个变量在SSE式子中同样需要惩罚，那么SSE就变成了下面这样：

$$\begin{aligned} SSE &= \frac{1}{2} \sum_{u,i} e_{ui}^2 + \frac{1}{2} \lambda \sum_u |p_u|^2 + \frac{1}{2} \lambda \sum_i |q_i|^2 + \frac{1}{2} \lambda \sum_u b_u^2 + \frac{1}{2} \lambda \sum_i b_i^2 \\ &= \frac{1}{2} \sum_{u,i} (r_{ui} - \mu - b_u - b_i - \sum_{k=1}^K p_{uk} q_{ki})^2 + \frac{1}{2} \lambda \sum_u |p_u|^2 + \frac{1}{2} \lambda \sum_i |q_i|^2 + \frac{1}{2} \lambda \sum_u b_u^2 + \frac{1}{2} \lambda \sum_i b_i^2 \end{aligned}$$

由上式可以看出SSE对Puk和qik的导数都没有变化，但此时多了bu和bi变量，同样要求出其更新式。首先求SSE对bu的导数，只有第一项和第四项和bu有关，第一项对bu的求导和之前的求导类似，用链式法则即可求得，第四项直接求导即可，最后可得偏导数为

$$\frac{\partial}{\partial b_u} SSE = -e_{ui} + \lambda b_u$$

同理可得对bi的导数为

$$\frac{\partial}{\partial b_i} SSE = -e_{ui} + \lambda b_i$$

所以往其负梯度方向变化得到其更新式为

$$\begin{aligned} b_u &:= b_u + \eta(e_{ui} - \lambda b_u) \\ b_i &:= b_i + \eta(e_{ui} - \lambda b_i) \end{aligned}$$

这就是修改后的SVD（RSVD）。



```
1 #coding:utf8
2 import numpy as np
3
4 class SVD():
5     def __init__(self,trainfile,testfile,factorNum=10):
6         self.trainfile=trainfile
7         self.testfile=testfile
8         self.factorNum=factorNum
9         self.userNum= 6040 # max user id
10        self.itemNum= 3952 # max movie id
11        self.learningRate=0.01
12        self.regularization=0.02
13        score=[float(line.split('::')[2])for line in open(self.trainfile)]
14        self.av= np.mean(score)
15        self.bu=np.zeros(self.userNum)
16        self.bi=np.zeros(self.itemNum)
17        temp=np.sqrt(self.factorNum)
18        self.pu=np.array([[0.1*np.random.random()/temp for i in range(self.factorNum)] for j in
range(self.userNum)])
19        self.qi=np.array([[0.1*np.random.random()/temp for i in range(self.factorNum)] for j in
range(self.itemNum)])
20
21    def train(self,iterTimes=100):
22        preRmse=10000.0
23        for iter in range(iterTimes):
24            fi=open(self.trainfile,'r')
25            for line in fi:
26                content=line.split('::')
27                user=int(content[0])-1
28                item=int(content[1])-1
29                rating=float(content[2])
30                pscore=self.predict(self.av,self.bu[user],self.bi[item],self.pu[user],self.qi[item])
31                eui=rating-pscore
32                self.bu[user]+=self.learningRate*(eui-self.regularization*self.bu[user])
```

```
33         self.bi[item]+=self.learningRate*(eui-self.regularization*self.bi[item])
34         temp=self.pu[user]
35         self.pu[user]+=self.learningRate*(eui*self.qi[item]-
self.regularization*self.pu[user])
36         self.qi[item]+=self.learningRate*(temp*eui-self.regularization*self.qi[item])
37         fi.close()
38         curRmse=self.test(self.av,self.bu,self.bi,self.pu,self.qi)
39         print "Iteration %d times,RMSE is : %f" % (iter+1,curRmse)
40         if curRmse>preRmse:
41             print "The best RMSE is : %f" % (preRmse)
42             break
43         else:
44             preRmse=curRmse
45
46     def test(self,av,bu,bi,pu,qi):
47         rmse=0.0
48         cnt=0
49         fi=open(self.testfile)
50         for line in fi:
51             cnt+=1
52             content=line.split('::')
53             user=int(content[0])-1
54             item=int(content[1])-1
55             score=float(content[2])
56             pscore=self.predict(av,bu[user],bi[item],pu[user],qi[item])
57             rmse+=(score-pscore)**2
58         fi.close()
59         return np.sqrt(rmse/cnt)
60
61     def predict(self,av,bu,bi,pu,qi):
62         pscore=av+bu+bi+np.dot(pu,qi)
63         if pscore<1:
64             pscore=1
65         elif pscore>5:
66             pscore=5
67         return pscore
68
69 if __name__=='__main__':
70     s=SVD("train.dat","test.dat")
71     s.train() # Iteration 24 times,RMSE is : 0.854743
```



```
1 # To convert train.dat and test.dat from ratings.dat.
2 import numpy as np
3 f=open("ratings.dat",'r')
4 a=[line for line in f]
5 n=len(a)
6 tn=int(n*0.8)
7 np.random.shuffle(a)
8 d=open("train.dat",'w')
9 [d.write(a[i]) for i in range(tn)]
10 d.close()
11 d=open("test.dat",'w')
12 [d.write(a[i]) for i in range(tn,n)]
13 d.close()
```



标签: [矩阵运算](#), [机器学习](#)

好文要顶

关注我

收藏该文





1357

关注 - 0

粉丝 - 15

+加关注

0

推荐

0


反对

« 上一篇: [PCA人脸识别](#)

» 下一篇: [PCA和Softmax分类比较—Mnist与人脸数据集](#)

posted on 2016-11-23 21:45 1357 阅读(749) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

 注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

- 【推荐】超50万VC++源码：大型组态工控、电力仿真CAD与GIS源码库！
- 【活动】申请成为华为云云享专家 尊享9大权益
- 【工具】SpreadJS纯前端表格控件，可嵌入应用开发的在线Excel
- 【腾讯云】拼团福利，AMD云服务器8元/月



相关博文：

- [hive1.2.1实战操作电影大数据！](#)
- [MovieLens电影数据分析](#)
- [推荐系统 矩阵分解 随机梯度下降算法的具体实现过程](#)
- [推荐系统个人理解（实践部分）](#)
- [《Python For Data Analysis》学习笔记-1](#)

最新新闻：

- [爸妈上网比你还秀，互联网与银发一族有哪些误会？](#)
- [起底FB雇佣“黑公关”：曾挑拨库克与特朗普维护高通](#)
- [DG辱华对话爆料者：Instagram一直在删我帖子](#)
- [董明珠并没有闯红灯被抓 是谁冤枉了她？](#)
- [谷歌申请“VR鞋子”专利 让你站着感觉就像在行走](#)
- » [更多新闻...](#)