Matrix Factorization Movie Recommender

实验要求：

根据一个涉及影评者及其几部影片评分情况的字典，实现collaborative filtering，并为Toby进行推荐。

核心思想：

使用collaborative filtering中的latent factor methods（潜在因子算法）。

这种算法的思想是这样：每个用户（user）都有自己的偏好，比如在电影领域，A喜欢带有超级英雄、动作、科幻等元素（latent factor）的电影，如果一部电影（item）带有这些元素，那么就将这部电影推荐给该用户，也就是用元素去连接用户和电影。每个人对不同的元素偏好不同，而每部电影包含的元素、所属的类型也不一样。根据矩阵分解（matrix factorization）的吧思想，我们希望能找到这样两个矩阵：

一，**用户-潜在因子矩阵Q。**表示不同的用户对于不用元素的偏好程度，数值越高表示越喜欢。如：

	动作	科幻	历史	谍战	超级英雄
大雄	0.7	0.9	0.2	0.7	0.9

二，**潜在因子-电影矩阵P。**表示每部电影含有各种元素的成分，或者说属于各种类型的程度。如：

	动作	科幻	历史	谍战	超级英雄
《美国队长2》	0.8	0.6	0.1	0.8	1.0

利用这两个矩阵，我们能得出大雄对《美国队长2》这部电影的喜欢程度：大雄对动作电影的偏好*《美国队长2》含有动作的成分+对科幻电影的偏好*《美国队长2》含有科幻的成分+对历史的偏好*《美国队长2》含有历史的成分+……

即：0.7 * 0.8 + 0.9 * 0.6 + 0.2 * 0.1 + 0.7 * 0.8 + 0.9 * 1 = 2.58

每个用户对每部电影都这样计算，可以得到不同用户对不同电影的评分矩阵R：

$$\tilde{R} = QP^T$$

回到本题中来，本题给出的是不同用户对不同电影的评分矩阵R，但是部分内容空缺（未评分），而我们要做的是根据提供的部分补全矩阵，完成对未评分电影的评分预测，实现推荐。

主要思路是先随机生成Q、P矩阵，然后计算与初始矩阵R的残差，利用SGD随机梯度下降迭代，逐渐缩小残差，最后得到优化的Q、P矩阵，并计算得到最终的推荐矩阵R'。

核心代码片段：

计算残差、随机梯度下降：

```
def SGD(R,P,Q,K,epoch=5000,alpha=0.0002,beta=0.02): # 矩阵分解，epoch：梯度下降次数；alpha：步长；beta：β。
    Q = Q.T # 矩阵的转置
    loss = []
    # 梯度下降
    for step in range(epoch):
        e = 0
        for i in range(len(R)): # 遍历行
            for j in range(len(R[i])): # 遍历列
                eij = R[i][j] - np.dot(P[i,:],Q[:,j]) # 求残差值，.dot表示矩阵相乘
                for k in range(K):
                    if R[i][j]>0:        #限制评分大于零
                        P[i][k] = P[i][k] + alpha*(2*eij*Q[k][j]-beta*P[i][k]) # 加入正则化，更新P
                        Q[k][j] = Q[k][j] + alpha*(2*eij*P[i][k]-beta*Q[k][j]) # 加入正则化，更新Q
                # 计算损失值
                if R[i][j] > 0:
                    e = e + pow(R[i][j] - np.dot(P[i, :], Q[:, j]), 2)  # 损失值的和
                    for k in range(K):
                        e = e + (beta / 2) * (pow(P[i][k], 2) + pow(Q[k][j], 2))  # 加入正则化后的损失值的和
        loss.append(e)
        if e<0.001: # 收敛条件，0.001为阈值
            break
    return P,Q.T,loss
```

绘制推荐指数表：

```
recTable = PrettyTable(['Lady in the Water','Snake on a Plane','Just My Luck',
                        'Superman Returns','You, Me and Dupree','The Night Listener'])
for i in range(7):
    recTable.add_row(R_MF[i])
recTable.add_column('Movies',['Lisa Rose','Gene Seymour','Michael Phillips',
                              'Claudia Puig','Mick LaSalle','Jack Matthews','Toby'])
print("推荐指数表：\n",recTable)
```

实验结果：

- 选取了迭代次数5000次，学习率0.0002下的一次结果：

用户评分表、矩阵Q、矩阵P、推荐矩阵：

用户评分表：

Lady in the Water	Snake on a Plane	Just My Luck	Superman Returns	You, Me and Dupree	The Night Listener	Movies
2.5	3.5	3	3.5	2.5	3	Lisa Rose
3	3.5	1.5	5	3.5	3	Gene Seymour
2.5	3	0	3.5	0	4	Michael Phillips
0	3.5	3	4	2.5	4.5	Claudia Puig
3	4	2	3	2	3	Mick LaSalle
3	4	0	5	3.5	3	Jack Matthews
0	4.5	0	4	1	0	Toby

```
矩阵Q:
 [[1.08587889 1.31052545]
 [1.29152931 1.75953942]
 [0.28847517 1.73869752]
 [1.97848125 1.41283335]
 [1.72393767 0.35094802]
 [0.69439863 2.22316084]]
矩阵P:
 [[0.99074427 1.21252044]
 [1.9649398  0.64598494]
 [0.56948312 1.5110804 ]
 [0.912884   1.5847376 ]
 [0.93501644 1.17292907]
 [1.89316638 0.80570129]
 [0.4033152  2.14809691]]
推荐矩阵:
 [[2.66486718 3.41305278 2.39401141 3.67325828 2.13351302 3.38359943]
 [2.98026635 3.67441332 1.69000875 4.80026562 3.61414089 2.80057992]
 [2.59869902 3.39430967 2.79159349 3.26161646 1.51206409 3.75482307]
 [3.06812042 3.96742472 3.01872371 4.04509401 2.12991564 4.15703197]
 [2.55246801 3.27141608 2.3090979  3.50706581 2.02354721 3.25688411]
 [3.11164146 3.86274306 1.94700234 4.88391585 3.54646012 3.1058157 ]
 [3.25308713 4.30055459 3.8512372  3.83285451 1.44916063 5.05562646]]
```

推荐指数表：给Toby的电影推荐指数：

推荐指数表：

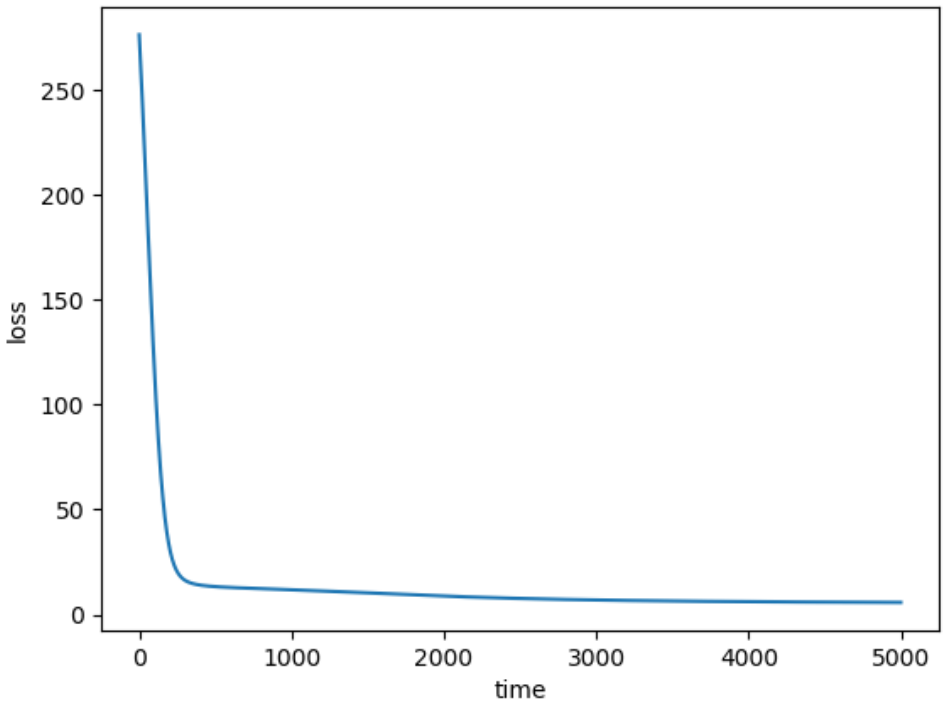
Lady in the Water	Snake on a Plane	Just My Luck	Superman Returns	You, Me and Dupree	The Night Listener	Movies
2.664867183235986	3.4130527789395844	2.3940114109059554	3.673258280687496	2.1335130235781654	3.3835994283177344	Lisa Rose
2.98026635106757	3.6744133155660115	1.6900087526922816	4.80026561968835	3.6141408861285216	2.800579919957641	Gene Seymour
2.598699015161321	3.394309667944537	2.7915934860417697	3.2616164610854255	1.5120640872866218	3.7548230679936534	Michael Phillips
3.0681204155255988	3.9674247170245796	3.0187237079020046	4.045094007217543	2.129915643051738	4.157031973936954	Claudia Puig
2.5524680119115164	3.271416077281278	2.309097896083191	3.5070658093775857	2.0235472097442795	3.2568841138041567	Mick LaSalle
3.111641458688327	3.862743063872485	1.9470023355366388	4.883915849427923	3.5464601242183007	3.1058157037936276	Jack Matthews
3.253087125968894	4.300554588074898	3.8512372018817254	3.8328545143301525	1.4491606289624215	5.0556264591955005	Toby

给Toby的电影推荐指数：

Lady in the Water	Just My Luck	The Night Listener
3.253087125968894	3.8512372018817254	5.0556264591955005

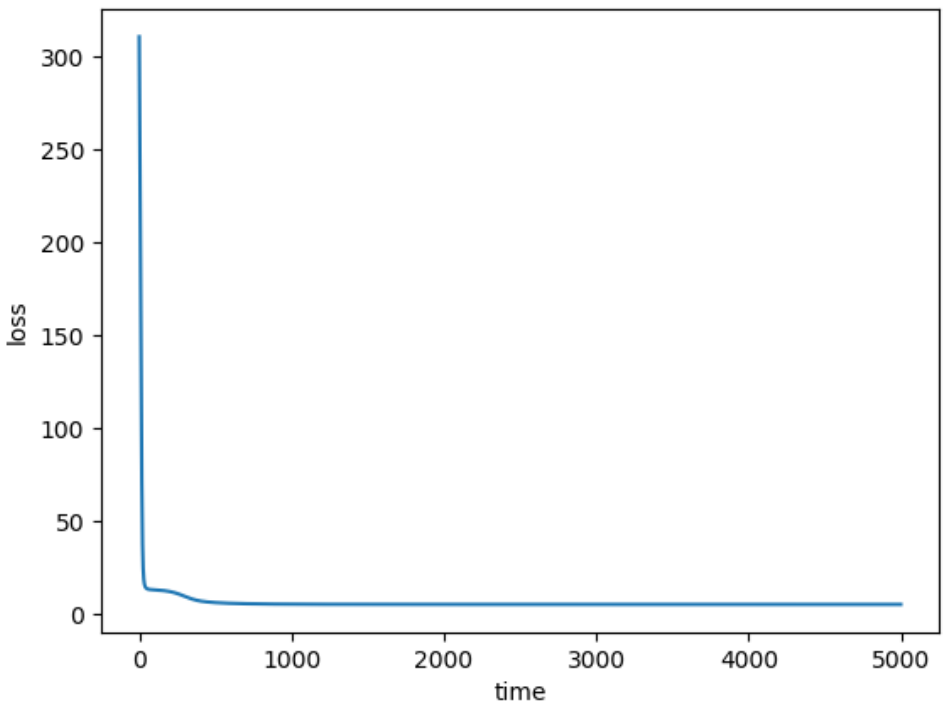
有最后一个表的推荐指数高低可以看出，对于用户用户Toby，首先推荐电影*The Night Listener*，其次是*Just My Luck*，最后是*Lady in the Winter*。

loss曲线：



- 又选取了迭代次数5000次，学习率为0.002下的一次结果：

```
给Toby的电影推荐指数:
 +-------------------+-------------------+--------------------+
| Lady in the Water |    Just My Luck   | The Night Listener |
+-------------------+-------------------+--------------------+
| 3.695892737270649 | 5.818011168416938 | 6.624901854093289  |
+-------------------+-------------------+--------------------+
```



多次改变迭代次数、学习率取值发现，推荐的结果（三部电影的推荐指数排序）基本不变，但若在迭代次数达上千次的情况下学习率取的过大，曲线会收敛过快，效果不是很好，有时可能会出现负值的过拟合现象。迭代次数5000次，学习率0.0002是一组较好的取值。

实验总结：

这次的实验用到了Collaborative Filtering的第二种方法：Latent Factor Methods。关于这个方法我之前存在一些误解，我之前认为由于第二种方法需要有每个样本的properties，再根据properties放到二维坐标系中去比较邻近关系，考虑到数据样本很小，加上这几个电影我们都不熟悉，更不知道它们的properties，如果强行加上properties很可能推荐结果较差，因此没有选择这种方法。后来进一步学习、了解后发现，我们不用对样本数据中的电影加上他们的property，这个只是这个方法的一种思路，我们可以随机生成P、Q矩阵，然后用随机梯度下降的优化方法最终确定他们的值，来得出结果。我也为自己之前天真的想法感到可笑。

这次实验另外的一个收获就是我接触到了python中表格的绘制，调用、学习了prettytable库；同时也温故而知新了numpy库的一些矩阵操作。