智能推荐系统第三次编程作业实验报告

----------隐语义模型

1. 算法简介

1.1 简介

1. 用隐语义模型来进行协同过滤的目标
    - 揭示隐藏的特征,这些特征能够解释为什么给出对应的预测评分。
    - 这类特征可能是无法直接用语言解释描述的,事实上我们并不需要知道。
2. 通过矩阵分解进行降维分析
    - 协同过滤算法非常依赖历史数据,而一般的推荐系统中,偏好数据又往往是稀疏的;这就需要对原始数据做降维处理
    - 分解之后的矩阵,就代表了用户和物品的隐藏特征
3. 隐语义模型实例
    - 基于概率的隐语义分析(pLSA)
    - 隐式迪利克雷分布模型(LDA)
    - 矩阵因子分解模型(基于奇异值分解的模型, SVD)

1.2 对于LFM的理解

- 我们可以认为，用户之所以给电影打出这样的分数，是有内在原因的，我们可以挖掘出影响用户打分的隐藏因素，进而根据未评分电影与这些隐藏因素的关联度，决定此未评分电影的预测评分
- 应该有一些隐藏的因素，影响用户的打分，比如电影:演员、题材、年代。甚至不一定是人直接可以理解的隐藏因子
- 找到隐藏因子，可以对user和item进行关联(找到是由于什么使得user喜欢/不喜欢此item,什么会决定user喜欢/不喜欢此item)，就可以推测用户是否会喜欢某一部未看过的电影
- 对于用户看过的电影，会有相应的打分，但一个用户不可能看过所有电影，对于用户没有看过的电影是没有评分的，因此用户评分矩阵大部分项都是空的，是一个稀疏矩阵

- 如果我们能够根据用户给已有电影的打分推测出用户会给没有看过的电影的打分，那么 就可以根据预测结果给用户推荐他可能打高分的电影

2. 核心代码注解

1.1 数据读取以及预处理

- 先读取数据，结果发现给的数据类型是这样的

```
# 使用read_csv来读取csv，默认分隔符为逗号
pd.read_csv('train.csv')


user_id business_id date    stars
0   A2JGzkvNjckSmps_4FbKWw  Xg5qEQiB-7L6kGJ5F4K3bQ  2014-03-18 01:14:10 5.0
1   rypcWiSNGMOsuWsiSLh9xA  4RoTEeqB_MNn6yaqZmlZHg  2015-08-29 18:32:15 4.0
2   DgkOWdoh7HPjhKQEPBU_jQ  ZOmf-3NN4Z59b2Fw6VAM7g  2015-09-14 16:33:03 3.0
3   FIk4lQQu1eTe2EpzQ4xhBA  HK2Ki-PvnNN-YMTlX1uSVA  2012-09-29 02:03:42 4.0
4   VizhcyMWWPz3UDXEBeix4w  UPIYuRaZvknINOd1w8kqRQ  2011-06-10 20:35:42 3.0
... ... ... ... ...
7927    1O638BDK_fWuxgTVJwff-A  ZIUs7gncPOXOOXr1ZYviAQ  2008-05-01 22:52:19 5.0
7928    rypcWiSNGMOsuWsiSLh9xA  skOstgY4NDJYOX1MbNJ3Pg  2016-07-13 00:18:24 4.0
7929    qibGLHABNReGeJr2w4_8yQ  LtNgP4FqXp5nMFOHErK8cw  2012-06-16 02:15:50 3.0
7930    1dWLN4Mr4hKhu8MQUCKqXQ  o597EK6uvR5RuPMZEwYCUg  2013-12-06 16:57:33 4.0
7931    pQdIIsbV5dGOVz_kWfMRTw  nzV7HlERvD_v1DEfL_oiGQ  2008-03-31 01:08:57 1.0
7932 rows × 4 columns
```

- 为了更好地使用这些数据，我把他转换成了一个矩阵，纵索引是user_id，横索引是business_id，内容（key值）是用户的打分item。

```
data_path = 'train.csv'
dtype = {"user_id":np.string_, "business_id":np.string_, "stars":np.float32}
# 加载数据，我们只用三列数据，分别是用户ID，商品ID，评分
ratings = pd.read_csv(data_path, dtype=dtype, usecols=[0,1,3])
ratings_matrix = ratings.pivot_table(index=["user_id"], columns=["business_id"], values="stars")
ratings_matrix
```

2.2 计算预测值

这一部分是核心部分，我用了一个函数实现，函数输入是用户的评分矩阵R，我需要对R进行矩阵因子分解。

- 假设用户物品评分矩阵为 R，现在有m个用户，n个物品
- 我们想要发现 k个隐类,我们的任务就是找到两个矩阵P和Q，使这两个矩阵的乘积近似等于R，即将用户物品评分矩阵R分解成为两个低维矩阵相乘:

$$\hat{R}_{m*n} = \hat{P}^{T}_{m*k} * Q_{m*n} \approx R$$

> 用户特征矩阵P乘以电影特征矩阵Q得到用户的评分矩阵，其中用户评分矩阵R中的元素就是矩阵P与矩阵Q中相应特征向量的点积，反映的是用户特征与物品特征的契合程度，故R中数字越大反映用户对物品喜好程度越大。

现实中矩阵分解得到的预测评分矩阵 $\hat{R}$,与原评分矩阵R在已知的评分项上可能会有误差，我的目标是找到一个最好的分解方式，让分解之后的预测评分矩阵总误差最小。在代码中，我使用了梯度下降的算法，不断试图减小损失值，使得结果最接近。

```
'''
@输入参数
R:M*N的评分矩阵
K:隐特征向量维度
max_iter:最大迭代次数
alpha:步长
lamda:正则化系数

@输出
分解之后的P、Q
P:初始化用户特征矩阵M*k
Q：初始化物品特征矩阵N*K
'''

#给定超参数

K= 5
max_iter = 500
alpha = 0.02
lamda = 0.01

#核心算法
def LFM_grad_desc(R,K,max_iter,alpha=0.0002,lamda = 0.002):
    #基本维度参数定义
    M = len(R)
    N = len(R[0])

    #P、Q初始值，随机生成
    P = np.random.rand(M,K)
    Q = np.random.rand(N,K)
    Q = Q.T

    #开始迭代
    for step in range(max_iter):
        print(step)
        #对所有的用户u、物品i做遍历，对应的特征向量Pu，Qi梯度下降
        for u in range(M):
            for i in range(N):
```

```
                #对于每一个大于0的评分，求出预测的评分误差
                if R[u][i] > 0:
                    eui = np.dot(P[u,:],Q[:,i]) - R[u][i]

                    #带入公式，按照梯度下降算法更新当前的Pu与Qi
                    for k in range(K):
                        P[u][k] = P[u][k] - alpha * (2 * eui * Q[k][i] + 2 * lamda * P[u][k])
                        Q[k][i] = Q[k][i] - alpha * (2 * eui * P[u][k] + 2 * lamda * Q[k][i])

        #u、i遍历完成，所有的特征向量更新完成，可以得到P、Q，可以计算预测评分矩阵
        predR = np.dot(P,Q)

        #计算当前损失函数
        cost = 0
        ci = 0
        for u in range(M):
            for i in range(N):
                if R[u][i] > 0:
                    cost += (np.dot(P[u,:],Q[:,i]) - R[u][i]) ** 2
                    #加上正则化项
                    for k in range(K):
                        cost += lamda * (P[u][k] ** 2 + Q[k][i] ** 2)
            #ci += 1
            #print("次数: ",ci,"cost:",cost)
        if cost < 0.001:
            break

    return P,Q.T,cost
```

下面就是调用函数直接yucejieguo。

```
#预测结果
P,Q,cost = LFM_grad_desc(R,K,max_iter,alpha,lamda)
'''
print(P)
print(Q)
print(cost)
print(R)
'''
predR = P.dot(Q.T)
#预测矩阵
predR
```

2.3 对test集的预测，以及后续处理

- 读取test集

```
data_path = 'test.csv' #读取test集
ratings2 = pd.read_csv(data_path, dtype=dtype, usecols=[0,1])
ratings2
```

- 将结果保存

```python
# 写入结果，保存在result.csv上
import csv

# 创建文件对象
f = open('result.csv','w',encoding='utf-8',newline='' "")

# 基于文件对象构建 csv写入对象
csv_writer = csv.writer(f)

# 构建列表头
csv_writer.writerow(["user_id","business_id","stars"])

# 写入csv文件内容
for row in ratings2.itertuples():
    uid = getattr(row, 'user_id')
    iid = getattr(row, 'business_id')
    outcome = predict(uid, iid,ratings_matrix, similarity)
    if outcome != -1:
        csv_writer.writerow([uid,iid,outcome])# 如果有预测值就输出
    else:
        csv_writer.writerow([uid,iid,ratings_matrix_mean[uid]]) #如果没有预测值就输出历史平均值
# 关闭文件
f.close()
```

3. 对结果的分析

3.1 运行时间问题

一开始我设置训练5000次，我自己试验了一些小的例子发现运行速度已经有好几秒了，果不其然运行这次实验的数据集的时候每训练一次都要花费很长时间，无奈只好降低训练次数。或许有更加快速的做法?

3.2 矩阵稀疏问题

很遗憾的是本次数据集转换成ratings_matrix出现了大片大片的'NaN'，说明矩阵过于稀疏。我认为矩阵太稀疏的话会对分解结果造成很大的影响，训练结果不容易收敛。

> 降低数据的稀疏性问题给推荐系统带来的影响，人们专门研究了很多的解决方法。其中从不同角度对用户和产品信息进行分析、处理、降低数据的稀疏程度。基于项目的协同过滤推荐、降维法、智能Agent方法可以在一定程度上缓解数据稀疏性问题。BP神经网络缓解协同过滤推荐算法的稀疏性问题，将奇异值分解应用到协同过滤推荐算法，通过奇异值分解算法得到低维正交矩阵，较好地解决了数据稀疏性问题，但是推荐的准确性会有一定的下降。在采用神经网络模型进行聚类处理的总思路下，通过寻找基于对象属性信息的项目类间隐性联系化解数据稀疏性对高维数据聚类的影响。此外，应用聚类算法是解决用户—项目评分矩阵稀疏性的比较有效的方法。

我对矩阵的分解等做了简单的尝试，但都以失败告终。

3.3 时间数据利用问题

本次实验并没有对训练样本中给出的时间进行任何处理，这是因为我并不知道应该怎么利用。但是关于时间的处理我有一定的构思。

- 时间离现在越近的评分，我们可以认为他比时间离现在更远的评分更加具有代表意义。所以可以设置一个和距现在时间长度正相关的衰减系数。
- test数据集并不是让你预测现在时间的评分，而是某个特定时间，所以可以根据时间衰减系数算出每一年的相似矩阵，根据test数据集里时间的年份来确定使用哪一年的相似矩阵。

4.总结

4.1 作业上交内容

- 实验报告

  > 智能推荐系统第三次次作业report_董辰尧.pdf

- 可运行的代码

  > LFM.ipynb

- 训练集，测试集

  > train.csv、test.csv为了确保提交的代码可运行

- 最终结果

  > result.csv

4.2 参考文献

https://blog.csdn.net/chaiqunxing51/article/details/50977195

> 稀疏矩阵处理

https://blog.csdn.net/weixin_44023658/article/details/106462356?utm_medium=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~default-17.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~default-17.control

> LFM入门

https://blog.csdn.net/pengchengliu/article/details/80932232

> 梯度下降算法推导